**CHAPTER-1**

**INTRODUCTION**

* 1. **Overview**

Disease diagnostic and treatment is usually very difficult and abstract level task. That’s why humans are good at it and machines fail. The major drawback of allopathic medicine is that, that there is symptomatic treatment but not a root cause treatment which results in temporary solution in most of the cases but not permanent solution especially for genetic diseases. To find a better solution, identifying protein interactions that are responsible for the diseases plays a pivotal role. There are many researches that has been conducted and is still being conducted to find best solution for this problem. Various techniques in data mining and machine learning have also been employed to find the diseases and the protein interactions that are responsible for the cause which has yielded different results. The perspective of medical science and data mining are used for discovering various sorts of metabolic syndromes. Data mining with classification plays a significant role in the prediction of genetic diseases and data investigation. Various methods have been used for knowledge abstraction by using known methods of data mining for prediction of genetic diseases using symptoms protein interactions. In this work, numerous readings have been carried out to produce a prediction model using not only distinct techniques but also by relating two or more techniques.

Our project aims at diagnosing genetic diseases using symptoms and identifying protein-based interactions, responsible for them thereby identifying the root cause for a disease rather than simple symptomatic solution.

We classifying diseases using various methods like K -Nearest Neighbor Algorithm (KNN) and Naïve Bayes (NB) and Artificial Neural Networks (ANN) and propose possible treatment using symptoms and disease diagnosed. The nature of genetic diseases is complex and hence, the diseases must be handled carefully. Not doing so may affect the treatment and can even cause death.

The proposed aid considers 20 diseases like carcinoma, Neoplasm, Heart valve disease etc. We generate results using K -Nearest Neighbor Algorithm (KNN) which produced good performance in predicting the protein interactions responsible for diseases identified using symptoms. This model achieves an accuracy of up to 100% considering 1 neighbor and 70% accuracy considering 2 neighbors. It is also 100% efficient in predicting the diseases using symptoms. The purpose of the project is to provide intelligent aid to participating physicians using machine learning techniques for identifying protein interactions that are responsible for causing genetic diseases. The results obtained shows that our proposed model has stronger capability to predict protein interactions responsible for diseases diagnosed compare to other models. It also contains the repository consisting of diseases, their symptoms, treatments and proteins responsible for diseases.

* 1. **Motivation**

Genetic disease identification and treatment is a complex task. Finding root cause i.e. finding protein interactions responsible for the disease is difficult and, in some cases, involves complicated painful procedures and is also a time taking research. In order to make life easier by the use of technology, data mining and machine learning is applied to make life easier.

* 1. **Problem Statement**

This aid implemented to provide most précised and accurate treatment for the symptoms provided by the user based on protein – protein interactions which are responsible for the disease and its symptoms.

* 1. **Existing System**

There exists no system which gives accurate information about the protein interactions responsible for genetic diseases. There is also no single platform which gives repository about symptoms, proteins responsible and treatment for the genetic diseases chosen in this project.

**1.5 Proposed System**

A huge dataset having multiple protein interactions and diseases caused by them would be taken and would be classified using classification algorithms namely KNN classification and Naïve Bayes classifier having disease name as class label. Based on symptoms, the proposed system is capable of classifying diseases and then the protein- protein interactions that are responsible for diseases. The proposed system can suggest treatment based on symptoms and disease identified. That is, the possible treatments are identified and the best one would be suggested by the model based on its knowledge acquisition and rendition.

Keeping in mind about the various techniques available such as machine learning and others. Chapter 1 has been exclusively utilized to understand the objective of protein - protein interaction using machine learning and its associated technologies, challenges and application

The survey of the supporting literature and the details of the supporting material are listed in Chapter 2, which provides an overview about various techniques such as OhmNet and the techniques used identify protein - protein interaction from the given data set**.**

             In all there are 9 chapters. The contents of the remaining chapters are as follows:

            Chapter 3 depicts the requirement specifications like functional, hardware and software requirements. It also includes advantages of using python and flask used for UI design**.**

            In Chapter 4,theproposed methodology is discussed.

            In Chapter 5,System design is discussed.

In Chapter 6, the implementation details of the project which includes details of algorithm and their implementation are also discussed in this chapter.

In Chapter 7, testing is discussed.

            Finally, in Chapter 8 and 9, experiments and results are discussed and finally the report is concluded and extension of the developed technique is discussed.

* + 1. **Advantages of Proposed System**
* Précised diagnose system with learning capability
* A detailed connectivity generated with proteins, disease symptoms and treatment**.**

**1.6 Summary**

The project uses classification namely the K-nearest neighbor and Naïve Bayes algorithm.

The research was carried out based on data available obtained from earlier researches. The analyses obtained by two different algorithms are based on different number of proteins having different number of diseases yielding to better accuracy.

**CHAPTER-2**

**LITERATURE SURVEY**

The following papers were referred to get in-depth knowledge about the research performed on protein interaction, methods used to achieve targets, their challenges faced, advantages and disadvantages and our scope of improvement. The summary is as follows:

1. **Large-scale analysis of disease pathways in the human interactome**

Discovering disease pathways, which can be deﬁned assets of proteins associated with a given disease, is an important problem that has the potential to provide clinically action able insights for disease diagnosis, prognosis, and treatment. Computational methods aid the discovery by relying on protein-protein interaction (PPI)networks. They start with a few known disease-associated proteins and aim to ﬁnd the rest of the pathway by exploring the PPI network around the known disease proteins. However, the success of such methods has been limited, and failure cases have not been well understood. Here we study the PPI network structure of 519 disease pathways. We ﬁnd that 90% of pathways do not correspond to single well-connected components in the PPI network. Instead, proteins associated with a single disease tend to for many separate connected components/regions in the network. We then evaluate state-of-the-art disease pathway discovery methods and show that their performance is especially poor on diseases with disconnected pathways. Thus, we conclude that network connectivity structure alone may not be suﬃcient for disease pathway discovery. However, we show that higher-order network structures, such as small subgraphs of the pathway, provide a promising direction for the development of new methods.

The overall goal of network biology is to develop approaches that use genomic and other network information to better understand human disease. Given the complexity of this goal, we focused on studying the PPI network structure of disease pathways, deﬁned through sets of proteins associated with diseases. We found that disease pathways are fragmented and sparsely embedded in the PPI network, and that spatial clustering of disease pathways within the PPI network is statistically insignificant. To better understand broad caveats of current methodology for disease protein discovery we evaluated the performance of leading methods and found that the assumptions do not fully capture PPI network structure. We showed, however, that there is detectable higher-order PPI network structure around disease proteins that can be leveraged to boost algorithm performance. These ﬁndings provide new insights into the disease pathway PPI network structure and can guide methodological advances in disease protein discovery.

1. **Predicting multicellular function through multi-layer tissue networks**

Motivation: Understanding functions of proteins in speciﬁc human tissues is essential for insights into disease diagnostics and therapeutics, yet prediction of tissue-speciﬁc cellular function remains a critical challenge for biomedicine. Results: Here, we present OhmNet, a hierarchy-aware unsupervised node feature learning approach for multi-layer networks. We build a multi-layer network, where each layer represents molecular interactions in a different human tissue. OhmNet then automatically learns a mapping of proteins, represented as nodes, to a neural embedding-based low-dimensional space of features. OhmNet encourages sharing of similar features among proteins with similar network neighborhoods and among proteins activated in similar tissues. The algorithm generalizes prior work, which generally ignores relationships between tissues, by modeling tissue organization with a rich multiscale tissue hierarchy. We use OhmNet to study multicellular function in a multi-layer protein interaction network of 107 human tissues. In 48 tissues with known tissue-speciﬁc cellular functions, OhmNet provides more accurate predictions of cellular function than alternative approaches, and also generates more accurate hypotheses about tissue-speciﬁc protein actions. We show that taking into account the tissue hierarchy leads to improved predictive power. Remarkably, we also demonstrate that it is possible to leverage the tissue hierarchy in order to effectively transfer cellular functions to a functionally uncharacterized tissue. Overall, OhmNet moves from ﬂat networks to multiscale models able to predict a range of phenotypes spanning cellular subsystems.

We presented OhmNet, an approach for unsupervised feature learning in multi-layer networks. We use OhmNet to learn state-of-theart task-independent protein features on a multi-layer network with 107 tissues. OhmNet models tissue interdependence up and down a tissue hierarchy spanning dozens of biological scales. The learned features achieve excellent accuracy on the cellular function prediction task, allow us to transfer functions to unannotated tissues, and provide insights into tissues.

1. **PGCN: Disease gene prioritization by disease and gene embedding through graph convolutional neural networks**

Motivation: Proper prioritization of candidate genes is essential to the genome-based diagnostics of a range of genetic diseases. However, it is a highly challenging task involving limited and noisy knowledge of genes, diseases and their associations. While a number of computational methods have been developed for the disease gene prioritization task, their performance is largely limited by manually crafted features, network topology, or pre-defined rules of data fusion. Results: Here, we propose a novel graph convolutional networkbased disease gene prioritization method, PGCN, through the systematic embedding of the heterogeneous network made by genes and diseases, as well as their individual features. The embedding learning model and the association prediction model are trained together in an end-to-end manner. We compared PGCN with five state-of-the-art methods on the Online Mendelian Inheritance in Man (OMIM) dataset for tasks to recover missing associations and discover associations between novel genes and diseases. Results show significant improvements of PGCN over the existing methods. We further demonstrate that our embedding has biological meaning and can capture functional groups of genes.

In this paper, we proposed a novel, unified framework for disease gene prioritization. Our method automatically learns the embedding of diseases and genes by systematically incorporating the topology of the heterogeneous network, the neighborhood of the diseases and genes, and the disease- and gene-specific information. The embeddings and the association prediction models are trained in an end-to-end manner. Extensive experiments demonstrate the power of our method on recovering missing associations, and on discovering associations for novel genes and/or diseases that are not seen in the training. Our framework is generic and can be readily applied to tackle other important problems in computational biology, such as drug disease association (Pushpakom et al., 2019) and homolog detection for protein structure prediction.

**CHAPTER-3**

**SYSTEM REQUIREMENTS**

## 3.1 Functional Requirements

## A function of software system is defined in functional requirement and the behavior of the system is evaluated when presented with specific inputs or conditions which may include calculations, data manipulation and processing and other specific functionality. The functional requirements of the project are one of the most important aspects in terms of entire mechanism of modules.

## The working project’s functional requirement are as follows:

* + - Given symptom as input, predicted genetic disease as output in the form of text
    - Given disease as input, predicted protein responsible for given disease as output in the form of text
    - Given disease as input, predicted treatment for given disease as output in the form of text
    - Cross verification for predicted protein.
    - Interactive Graphical user interface.
    - User interface should provide text fields to input symptoms.
    - Based on the symptoms, disease identified and respective best treatment should be displayed in graph or text form as output.

## 3.2 Non-Functional Requirements

Nonfunctional requirements describe how a system must behave and establish constraints of its functionality. This type of requirements is also known as the system’s *quality attributes*. Attributes such as performance, security, usability, compatibility are not the feature of the system, they are a required characteristic. They are "developing" properties that emerge from the whole arrangement and hence we can't compose a particular line of code to execute them. Any attributes required by the customer are described by the specification. We must include only those requirements that are appropriate for our project.

Non-Functional Requirements are as follows:

* **Reliability**

The structure must be reliable and strong in giving the functionalities. The movements must be made unmistakable by the structure when a customer has revealed a couple of enhancements. When the doctor is giving an input to cross verify his results, the proposed output should be reliable.

* **Maintainability**

The system watching and upkeep should be fundamental and focus in its approach. There should not be an excess of occupations running on diverse machines such that it gets hard to screen whether the employments are running without lapses. This system should be easily maintainable by every user.

* **Performance**

The framework will be utilized by numerous representatives all the while. It should allow brisk accessibility to each and every piece of its users. For instance, if a doctor gives symptoms as an input, the speed and accuracy of the model should be fast and approaching.

* **Portability**

The framework should to be effectively versatile to another framework. This is obliged when the web server, which is facilitating the framework gets adhered because of a few issues, which requires the framework to be taken to another framework.

* **Scalability**

The framework should be sufficiently adaptable to include new functionalities at a later stage. As and when number of diseases increases, the model should adapt to the changes and always perform at its best.

* **Flexibility**

Flexibility is the capacity of a framework to adjust to changing situations and circumstances, and to adapt to changes to business approaches and rules.

**3.3 Software Requirements**

* **Operating system** : Windows 8 / 10
* **Programming Language** : Python
* **Framework** : Anaconda
* **IDE** :Jupyter Notebook/spyder
* **DL Libraries** : Numpy, Pandas
* **UI** : Flask framework

**3.3.1 Python**

**Python** is an [interpreted](https://en.wikipedia.org/wiki/Interpreted_language), [high-level](https://en.wikipedia.org/wiki/High-level_programming_language), [general-purpose](https://en.wikipedia.org/wiki/General-purpose_programming_language) [programming language](https://en.wikipedia.org/wiki/Programming_language). Its language constructs and [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) approach aims to help programmers write clear, logical code for small and large-scale projects. Python is [dynamically typed](https://en.wikipedia.org/wiki/Dynamic_programming_language) and [garbage-collected](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)). It supports multiple [programming paradigms](https://en.wikipedia.org/wiki/Programming_paradigm), including [procedural](https://en.wikipedia.org/wiki/Procedural_programming), object-oriented, and [functional programming](https://en.wikipedia.org/wiki/Functional_programming). Python is often described as a "batteries included" language due to its comprehensive [standard library](https://en.wikipedia.org/wiki/Standard_library). Python [interpreters](https://en.wikipedia.org/wiki/Interpreter_(computing)) are available for many [operating systems](https://en.wikipedia.org/wiki/Operating_system). A global community of programmers develops and maintains [CPython](https://en.wikipedia.org/wiki/CPython), an [open source](https://en.wikipedia.org/wiki/Open-source_software) [reference implementation](https://en.wikipedia.org/wiki/Reference_implementation). Python is a [multi-paradigm programming language](https://en.wikipedia.org/wiki/Multi-paradigm_programming_language). [Object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming) and [structured programming](https://en.wikipedia.org/wiki/Structured_programming) are fully supported, and many of its features support [functional programming](https://en.wikipedia.org/wiki/Functional_programming) and [aspect-oriented programming](https://en.wikipedia.org/wiki/Aspect-oriented_programming) (including by [metaprogramming](https://en.wikipedia.org/wiki/Metaprogramming) and [metaobjects](https://en.wikipedia.org/wiki/Metaobject) (magic methods)). Many other paradigms are supported via extensions, including [design by contract](https://en.wikipedia.org/wiki/Design_by_contract) and [logic programming](https://en.wikipedia.org/wiki/Logic_programming). Python uses [dynamic typing](https://en.wikipedia.org/wiki/Dynamic_typing), and a combination of [reference counting](https://en.wikipedia.org/wiki/Reference_counting) and a cycle-detecting garbage collector for [memory management](https://en.wikipedia.org/wiki/Memory_management). It also features dynamic [name resolution](https://en.wikipedia.org/wiki/Name_resolution_(programming_languages)) ([late binding](https://en.wikipedia.org/wiki/Late_binding)), which binds method and variable names during program execution. Rather than having all of its functionality built into its core, Python was designed to be highly [extensible](https://en.wikipedia.org/wiki/Extensibility). This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Python strives for a simpler, less-cluttered syntax and grammar while giving developers a choice in their coding methodology. Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use [curly brackets](https://en.wikipedia.org/wiki/Curly_bracket_programming_language) to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases. Python uses [whitespace](https://en.wikipedia.org/wiki/Whitespace_character) indentation, rather than [curly brackets](https://en.wikipedia.org/wiki/Curly_bracket_programming_language) or keywords, to delimit [blocks](https://en.wikipedia.org/wiki/Block_(programming)). An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block.

Thus, the program's visual structure accurately represents the program's semantic structure. This feature is also sometimes termed the [off-side rule](https://en.wikipedia.org/wiki/Off-side_rule). Python's large [standard library](https://en.wikipedia.org/wiki/Standard_library), commonly cited as one of its greatest strengths, provides tools suited too many tasks. For Internet-facing applications, many standard formats and protocols such as [MIME](https://en.wikipedia.org/wiki/MIME) and [HTTP](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol) are supported. It includes modules for creating [graphical user interfaces](https://en.wikipedia.org/wiki/Graphical_user_interface), connecting to [relational databases](https://en.wikipedia.org/wiki/Relational_database), [generating pseudorandom numbers](https://en.wikipedia.org/wiki/Pseudorandom_number_generator), arithmetic with arbitrary precision decimals, manipulating [regular expressions](https://en.wikipedia.org/wiki/Regular_expression), and [unit testing](https://en.wikipedia.org/wiki/Unit_testing). The [Python Package Index](https://en.wikipedia.org/wiki/Python_Package_Index) (PyPI), the official repository for third-party Python software, contains packages with a wide range of functionality, including:

* Graphical user interfaces
* Web frameworks
* Multimedia
* Databases
* Networking
* Test frameworks
* Automation
* Web scraping
* Documentation
* System administration
* Scientific computing
* Text processing
* Image processing

Python can serve as a [scripting language](https://en.wikipedia.org/wiki/Scripting_language) for [web applications](https://en.wikipedia.org/wiki/Web_application). Libraries such as [NumPy](https://en.wikipedia.org/wiki/NumPy), [SciPy](https://en.wikipedia.org/wiki/SciPy) and [Matplotlib](https://en.wikipedia.org/wiki/Matplotlib) allow the effective use of Python in [scientific computing](https://en.wikipedia.org/wiki/Scientific_computing), with specialized libraries such as [Biopython](https://en.wikipedia.org/wiki/Biopython) and [Astropy](https://en.wikipedia.org/wiki/Astropy) providing domain-specific functionality. Python is commonly used in [artificial intelligence](https://en.wikipedia.org/wiki/Artificial_intelligence) projects with the help of libraries like [TensorFlow](https://en.wikipedia.org/wiki/TensorFlow), [Keras](https://en.wikipedia.org/wiki/Keras) and [Scikit-learn](https://en.wikipedia.org/wiki/Scikit-learn). As a scripting language with [modular architecture](https://en.wikipedia.org/wiki/Modular_programming), simple syntax and rich text processing tools, Python is often used for [natural language processing](https://en.wikipedia.org/wiki/Natural_language_processing). Python is used extensively in the [information security](https://en.wikipedia.org/wiki/Information_security) industry, including in exploit development.

* **Python is Interpreted −** Python is processed at run-time by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
* **Python is Interactive −** you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
* **Python is Object-Oriented −** Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
* **Python is a Beginner's Language −** Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

**Features:**

Python's features include –

* **Easy-to-maintain −** Python's source code is fairly easy-to-maintain.
* **A broad standard library −** Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
* **Interactive Mode −** Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
* **Extendable −** you can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
* **Databases −** Python provides interfaces to all major commercial databases.
* **GUI Programming−** Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
* **Easy to learn:**

The most alluring factor of Python is that anyone aspiring to learn this language can learn it easily and quickly. When compared to other data science languages like R, Python promotes a shorter learning curve and scores over others by promoting an easy-to-understand syntax.

* **Scalability:**

When compared to other languages like R, Python has established a lead by emerging as a scalable language, and it is faster than other languages like Matlab and Stata. Python’s scalability lies in the flexibility that it gives to solve problems, as in the case of YouTube that migrated to Python. Python has come good for different usages in different industries and for rapid development of applications of all kinds.

**3.3.2 Packages**

**3.3.2.1 Numpy**

**NumPy is a library for the** [**Python programming language**](https://en.wikipedia.org/wiki/Python_(programming_language))**, adding support for large, multi-dimensional** [**arrays**](https://en.wikipedia.org/wiki/Array_data_structure) **and** [**matrices**](https://en.wikipedia.org/wiki/Matrix_(math))**, along with a large collection of** [**high-level**](https://en.wikipedia.org/wiki/High-level_programming_language)[**mathematical**](https://en.wikipedia.org/wiki/Mathematics)[**functions**](https://en.wikipedia.org/wiki/Function_(mathematics)) **to operate on these arrays. NumPy is** [**open-source software**](https://en.wikipedia.org/wiki/Open-source_software) **and has many contributors. The core functionality of NumPy is its "ndarray", for *n*-dimensional array, data structure. These arrays are** [**strided**](https://en.wikipedia.org/wiki/Stride_of_an_array) **views on memory. In contrast to Python's built-in list data structure (which, despite the name, is a** [**dynamic array**](https://en.wikipedia.org/wiki/Dynamic_array)**), these arrays are homogeneously typed: all elements of a single array must be of the same type. NumPy has built-in support for** [**memory-mapped**](https://en.wikipedia.org/wiki/Memory-mapped_file) **ndarrays. A new package called *Numarray* was written as a more flexible replacement for Numeric. Like Numeric, it is now deprecated. Numarray had faster operations for large arrays, but was slower than Numeric on small ones, so for a time both packages were used for different use cases. We can import numpy as follows:**

**>>> import numpy as np**

**3.3.2.2 Pandas**

When you want to use Pandas for data analysis, you’ll usually use it in one of three different ways:

* Convert a Python’s list, dictionary or Numpy array to a Pandas data frame
* Open a local file using Pandas, usually a CSV file, but could also be a delimited text file (like TSV), Excel, etc
* Open a remote file or database like a CSV or a JSONon a website through a URL or read from a SQL table/database

There are different commands to each of these options.

One of the things that is so much easier in Pandas is selecting the data you want in comparison to selecting a value from a list or a dictionary.

The last set of basic Pandas commands are for joining or combining data frames or rows/columns.

Pandas is a [Python](https://www.python.org/) package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way toward this goal.

Pandas is well suited for many different kinds of data:

* Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
* Ordered and unordered (not necessarily fixed-frequency) time series data.
* Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels
* Any other form of observational / statistical data sets. The data actually need not be labeled at all to be placed into a pandas data structure

The two primary data structures of pandas, [Series](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html#pandas.Series) (1-dimensional) and [Data Frame](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html#pandas.DataFrame) (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering. For R users, [DataFrame](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html#pandas.DataFrame) provides everything that R’s data.frame provides and much more. Pandas are built on top of [NumPy](https://www.numpy.org/) and is intended to integrate well within a scientific computing environment with many other 3rd party libraries.

Here are just a few of the things that pandas does well:

* Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data
* Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects
* Automatic and explicit data alignment: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let Series, DataFrame, etc. automatically align the data for you in computations
* Powerful, flexible group by functionality to perform split-apply-combine operations on data sets, for both aggregating and transforming data
* Make it easy to convert ragged, differently-indexed data in other Python and NumPy data structures into DataFrame objects
* Intelligent label-based slicing, fancy indexing, and subsetting of large data sets
* Intuitive merging and joining data sets
* Flexible reshaping and pivoting of data sets
* Hierarchical labeling of axes (possible to have multiple labels per tick)
* Robust IO tools for loading data from flat files (CSV and delimited), Excel files, databases, and saving / loading data from the ultrafast HDF5 format
* Time series-specific functionality: date range generation and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging, etc.

## We import Pandas using,

>>> import pandas as pd

* + 1. **Flask**

**Flask** is a micro [web framework](https://en.wikipedia.org/wiki/Web_framework) written in [Python](https://en.wikipedia.org/wiki/Python_(programming_language)). It is classified as a [microframework](https://en.wikipedia.org/wiki/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. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools. Extensions are updated far more frequently than the core Flask program.

Applications that use the Flask framework include [Pinterest](https://en.wikipedia.org/wiki/Pinterest) and [LinkedIn](https://en.wikipedia.org/wiki/LinkedIn)

The microframework Flask is based on the *Pocoo* projects *Werkzeug* and *Jinja2*.

* **Werkzeug**

Werkzeug is a utility library for the [Python programming language](https://en.wikipedia.org/wiki/Python_(programming_language)), in other words a toolkit for [Web Server Gateway Interface](https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface) (WSGI) applications, and is licensed under a [BSD License](https://en.wikipedia.org/wiki/BSD_licenses). Werkzeug can realize software objects for request, response, and utility functions. It can be used to build a custom [software framework](https://en.wikipedia.org/wiki/Software_framework) on top of it and supports Python 2.6, 2.7 and 3.3.

* **Jinja**

Jinja, also by Ronacher, is a [template engine](https://en.wikipedia.org/wiki/Template_engine_(web)) for the Python programming language and is licensed under a BSD License. Similar to the [Django web framework](https://en.wikipedia.org/wiki/Django_(web_framework)), it handles templates in a [sandbox](https://en.wikipedia.org/wiki/Sandbox_(computer_security))

**Features of flask are as follows:**

* Development server and debugger
* Integrated support for unit testing
* RESTful request dispatching
* Uses [Jinja](https://en.wikipedia.org/wiki/Jinja_(template_engine)) templating
* Support for secure cookies (clientside sessions)
* 100% [WSGI](https://en.wikipedia.org/wiki/WSGI) 1.0 compliant
* [Unicode](https://en.wikipedia.org/wiki/Unicode)-based
* Extensive documentation
* [Google App Engine](https://en.wikipedia.org/wiki/Google_App_Engine) compatibility
* Extensions available to enhance features desired

**Estimators**

The estimator interface is at the core of the library. It defines instantiation mechanisms of objects and exposes a fit method for learning a model from training data. All supervised and unsupervised learning algorithms (e.g., for classification, regression or clustering) are offered as objects implementing this interface. Machine learning tasks like feature extraction, feature selection or dimensionality reduction are also provided as estimators.

**Predictor**

The predictor interface extends the notion of an estimator by adding a predict method that takes an array X test and produces predictions for X test, based on the learned parameters of the estimator (we call the input to predict “X test” in order to emphasize that predict generalizes to new data). In the case of supervised learning estimators, this method typically returns the predicted labels or values computed by the model.

**Transformers**

Since it is common to modify or filter data before feeding it to a learning algorithm, some estimators in the library implement a transformer interface which defines a transform method. It takes as input some new data X test and yields as output a **transformed version of X test.** Preprocessing, feature selection, feature extraction and dimensionality reduction algorithms are all provided as transformers within the library.

**Advanced API**

The advanced API mechanisms for building meta-estimators, composing complex estimators and selecting models

**Meta-estimators**

Some machine learning algorithms are expressed naturally as meta-algorithms parameterized on simpler algorithms. Examples include ensemble methods which build and combine several simpler models (e.g., decision trees), or multiclass and multi label classification schemes which can be used to turn a binary classifier into a multiclass or multi label classifier.

**3.4 Hardware Requirements**

* System Processor : Core i5
* Hard Disk : 1 TB.
* Ram : 8 GB.

Any desktop / Laptop system with above configuration or higher level

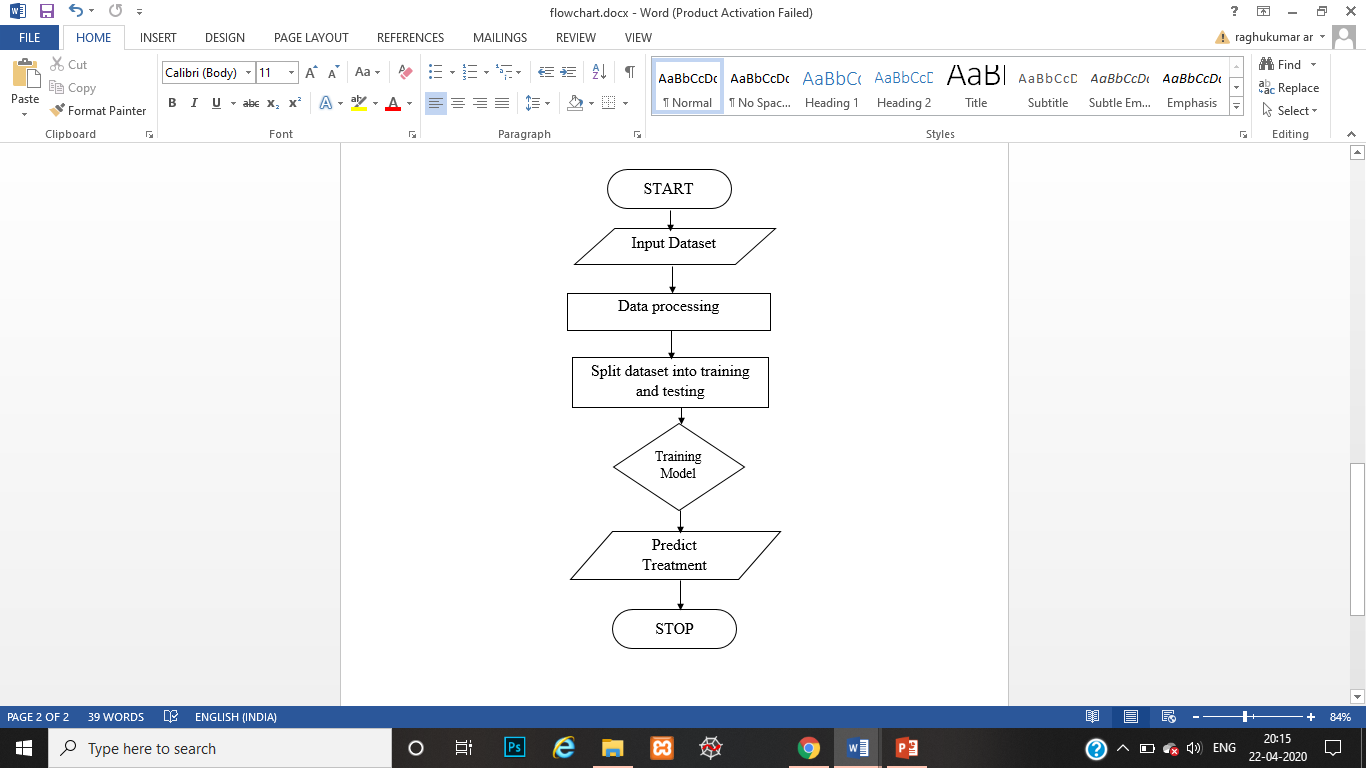
**CHAPTER-4**

**METHODOLOGY**

**4.1 Flowchart**

A [flowchart](https://project-management-knowledge.com/definitions/f/flowchart/) is one of the seven basic quality tools used in project management and it displays the actions that are necessary to meet the goals of a particular task in the most practical sequence.

Treatment Predictor is the ability to predict the treatment that has been provided to the system. For treatment prediction, we need to implement the naïve Byes Classifier.



**Fig 4.1: Flowchart1**



**Fig 4.1.a: Flowchart2**

1. Symptoms are taken as input and disease is predicted
2. Same disease is given as input and protein id causing that disease is predicted.
3. For cross verification same protein id is given input and disease is predicted.
4. The previously got output i.e., disease name is given as input and treatment is predicted.­­

**4.2 Data Collection**

In data collection process we collected the data from <https://snap.stanford.edu/data/> this website. Initially we took protein-protein interaction and Disease gene association network dataset. A protein-protein interaction data consists of 21,000 proteins having 3, 21,000 interactions. A Disease gene dataset consists of proteins that causes the disease. By considering disease gene dataset, manually symptoms-disease, disease-protein, protein-disease and disease-treatment datasets are generated. From the above dataset 20 diseases are considered they are:

|  |  |
| --- | --- |
| Carcinoma | Neoplasm |
| Squamous cell neoplasm | Ovarian diseases |
| Liver carcinoma | IGA |
| Obesity | Autistic Disorder |
| Kidney Failure | Brain neoplasm |
| Stomach neoplasm | prostatic neoplasm |
| Lymphoma | heart diseases |
| Schizophrenia | Peripheral neuropathy |
| Rheumatoid Arthritis | Adenoid Cystic Carcinoma |
| salivary gland neoplasm | Diabetic Mellitus |

Table 1: List of Diseases

**4.3 Data Pre-Processing**

In this step we process the data. Initially we check for the null value in the data set, if there is any null value replace it with the mean value of that parameter and then perform the null value analysis (contains 0 as value) and replace those null values with their respective mean value. Perform a regular expression check for the data set to convert it to the correct form.

**4.4 Data Splitting**

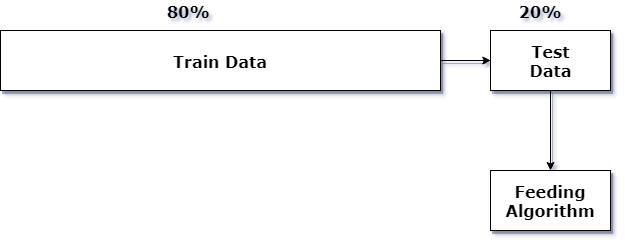
In this stage we perform the splitting of pre-processed data into test and train data. The proportion of training and a test set is usually 80 to 20 percent respectively. A training set is then split again, and its 20 percent will be used to form a validation set.

**4.4.1 Train data set**

Atraining data set is used to train a model and define its optimal parameters that it has to learn from data.

**4.4.2 Test data set**

A test set is needed for an evaluation of the trained model and its capability for generalization. The latter means a model’s ability to identify patterns in new unseen data after having been trained over a training data. It is shown in figure 4.2.



**Fig 4.2**: **Representation of data splitting**

**4.5 Constructing the train model**

After pre-processing the collected data and split it into two subsets, we can proceed with model training. This process entails “feeding” the algorithm with training data. An algorithm will process data and output a model that is able to find a target value (attribute) in new data — an answer you want to get with predictive analysis. The purpose of model training is to develop a model.

Two model training styles are most common — supervised and unsupervised learning. The choice of each style depends on whether you must forecast specific attributes or group data objects by similarities.

**Supervised learning:** Supervised learning allows for processing data with target attributes or labeled data. These attributes are mapped in historical data before the training begins. With supervised learning, we can solve classification and regression problems.

**Unsupervised learning:** During this training style, an algorithm analyzes unlabeled data. The goal of model training is to find hidden interconnections between data objects and structure objects by similarities or differences. Unsupervised learning aims at solving such problems as clustering, association rule learning, and dimensionality reduction. For instance, it can be applied at the data pre-processing stage to reduce data complexity.

**4.6 Summary**

This chapter provides the description on the methodology of the proposed system where the pre-processing of the given data is done accordingly the data analysis is performed.

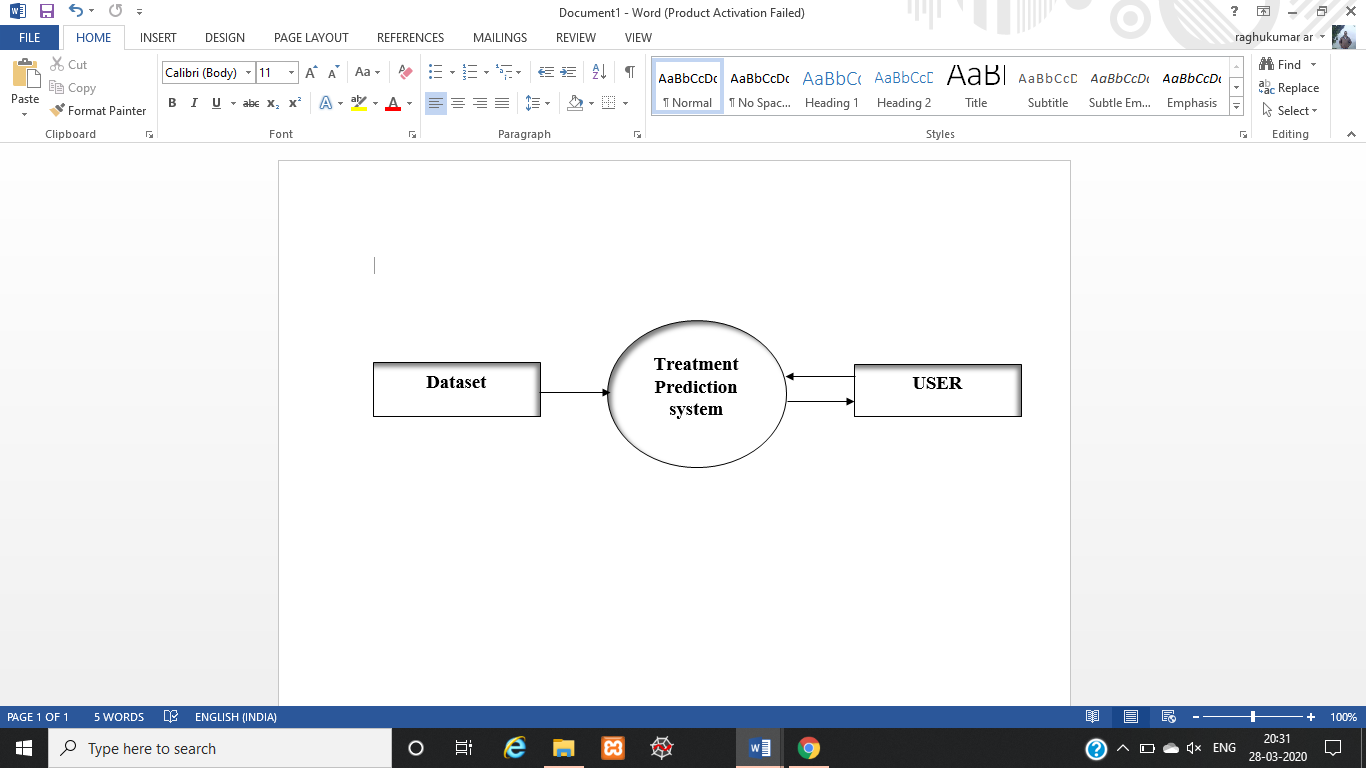
**CHAPTER-5**

**SYSTEM DESIGN**

**5.1 Data Flow Diagrams**

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects. A DFD is often used as a preliminary step to create an overview of the system without going into great detail, which can later be elaborated.

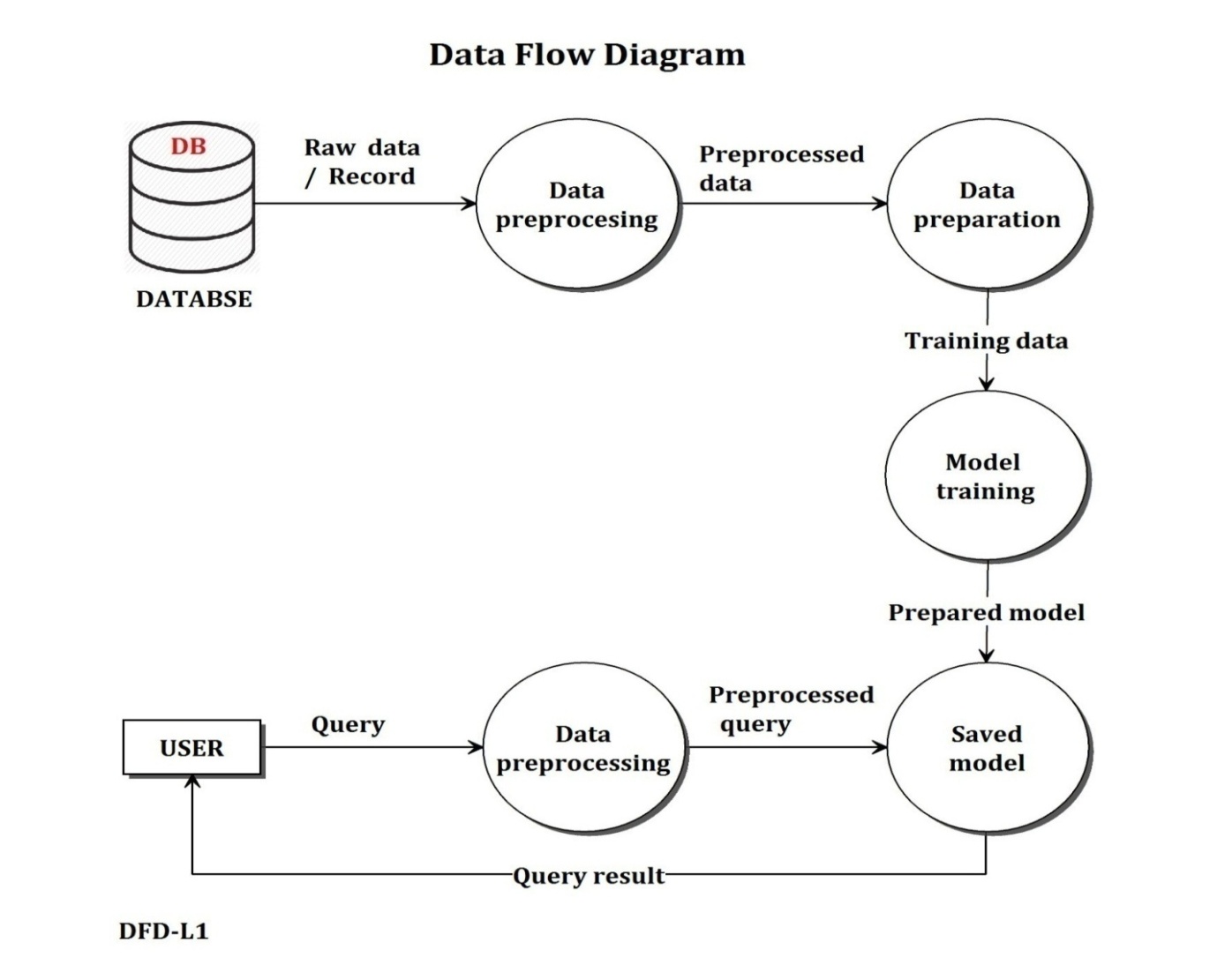
**DATAFLOW DIAGRAM LEVEL 0:**



**Fig 5.1: Dataflow diagram level 0**

This is the Zero level Data Flow Diagram of Disease diagnostic aid, where we have elaborated the high level process of treatment. It’s a basic overview of the whole disease diagnostic aid or process being analyzed. It’s designed to be at-a-glance view of user showing the system as a single high-level process.

**DATAFLOW DIAGRAM LEVEL 1:**

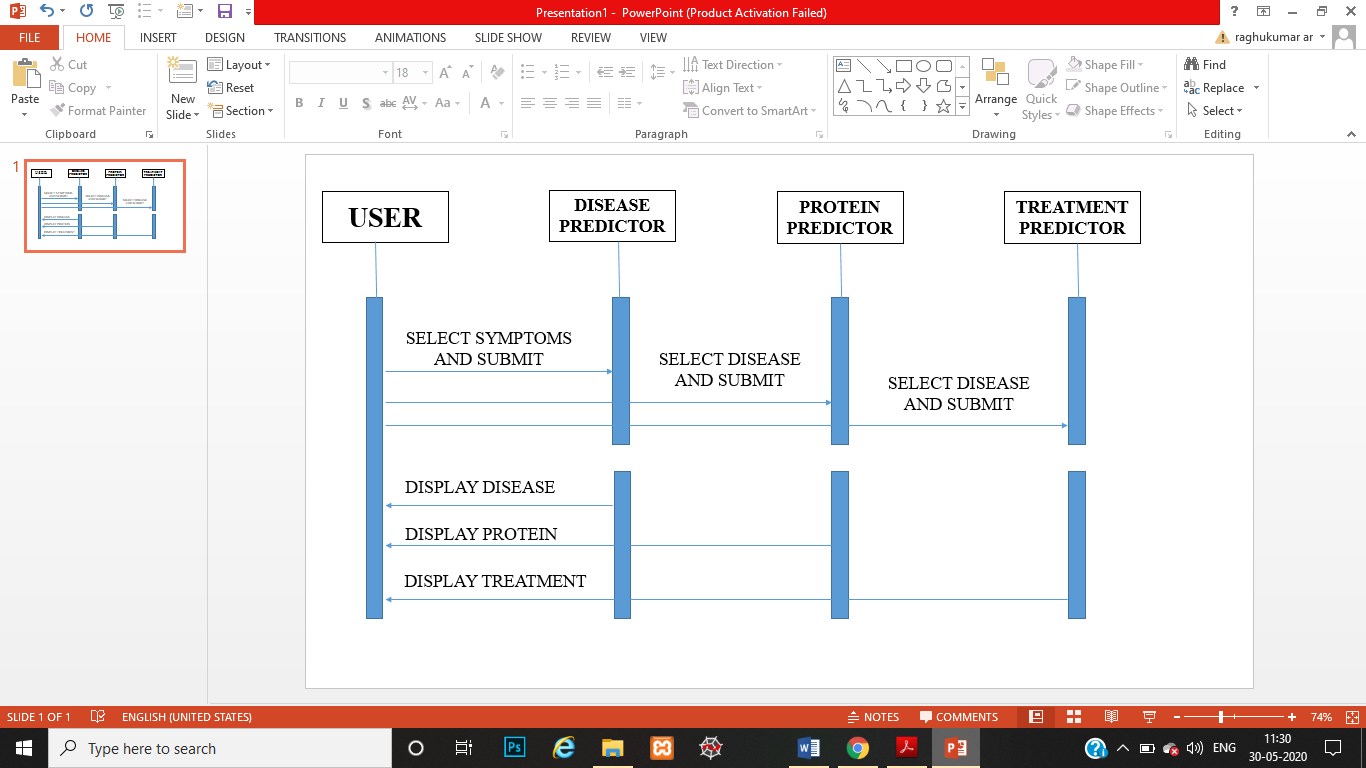


**Fig 5.2: Dataflow diagram level 1**

First level data flow diagram (1st level DFD) of treatment prediction system shows how the system is divided into sub-systems(processes), each of which deals with one or more of the data flows to or from an external user, and which together provide all of the functionality of the treatment prediction as a whole. DFD level 1 provides a more detailed breakout of pieces of the 1st level data flow diagram.

**5.2 SEQUENCE** **DIAGRAM**

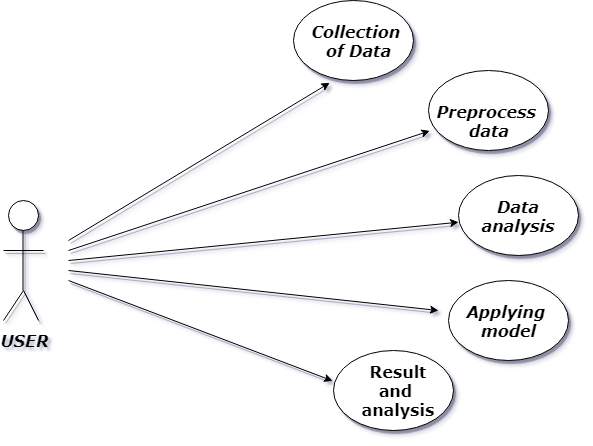
It explains the sequence of the Treatment Predictor. Initially system shows the symptoms to be selected. The user selects the symptoms and submits to the system. The Disease is predicted and display the result. Again the user selects the diseases and submits to the system, protein id which is responsible for this disease is displayed as result. Then the disease which is predicted as output previously is now taken as input here and for that disease treatment is predicted as output.



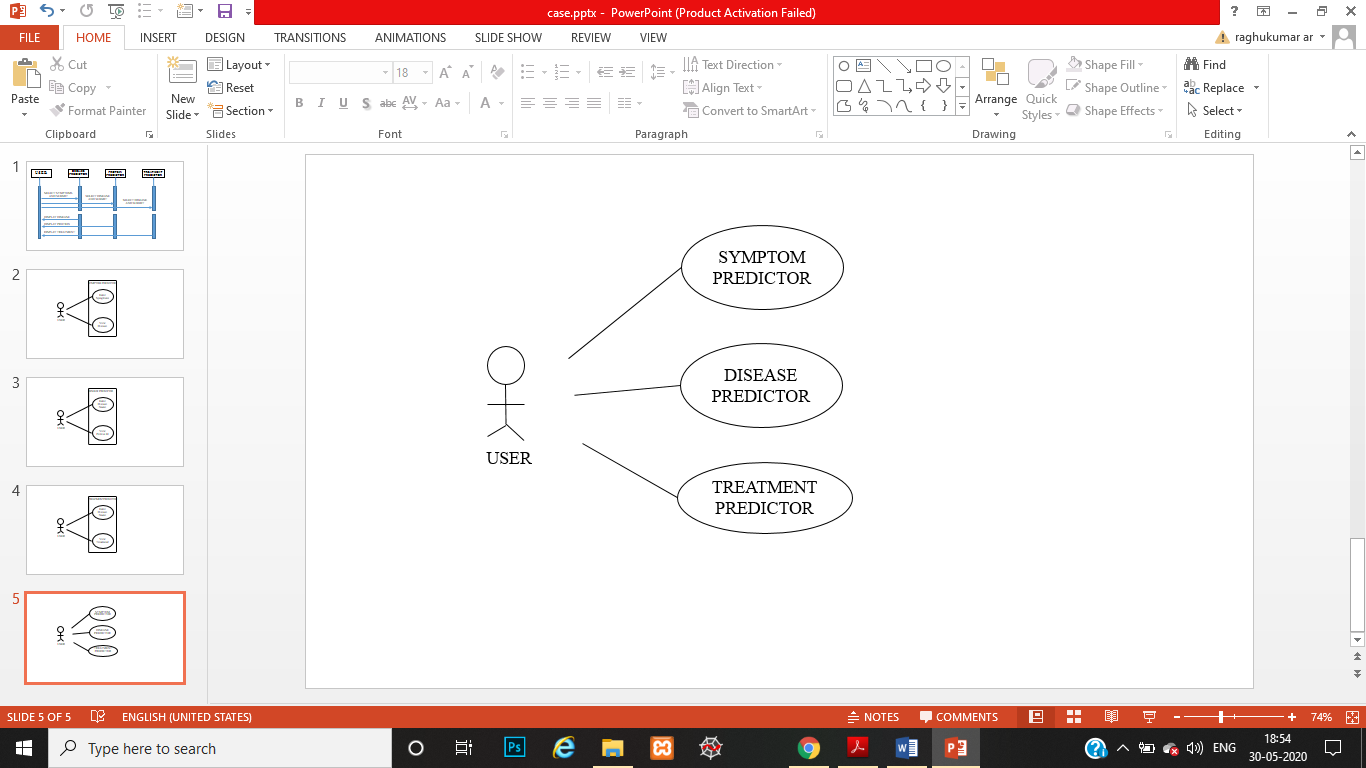
**Fig 5.3: Sequence** **diagram**

**5.3 USE CASE DIAGRAM**

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. In software and systems engineering, a use case is a list of actions or event steps, typically defining the interactions between a role (known in the Unified Modeling Language as an actor) and a system, to achieve a goal. The actor can be a human, an external system, or time. In systems engineering, use cases are used at a higher level than within software engineering, often representing missions or stakeholder goals. Another way to look at it is a use case describes a way in which a real-world actor interacts with the system. In a system use case you include high-level implementation decisions. System use cases can be written in both an informal manner and a formal manner.

****

**Fig 5.4: Use case Diagram**

****

**Fig 5.5: Use case Diagram for our model**

**CHAPTER-6**

**IMPLEMENTATION**

**6.1 K-nearest neighbor Algorithm**

K-Nearest Neighbors (KNN) is one of the simplest algorithms used in [Machine Learning for regression](https://quantra.quantinsti.com/course/trading-with-machine-learning-regression) and classification problem. KNN algorithms use data and classify new data points based on similarity measures (e.g. distance function). Classification is done by a majority vote to its neighbors. The data is assigned to the class which has the nearest neighbors. As you increase the number of nearest neighbors, the value of k, accuracy might increase.

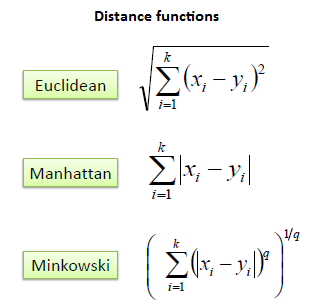
Algorithm:

1. For implementing any algorithm, we need dataset. So during the first step of KNN, we must load the training as well as test data.
2. Next, we need to choose the value of K i.e. the nearest data points. K can be any integer.
3. For each point in the test data do the following –

* Calculate the distance between test data and each row of training data with the help of any of the method namely: Euclidean, Manhattan or Hamming distance. The most commonly used method to calculate distance is Euclidean.
* Now, based on the distance value, sort them in ascending order.
* Next, it will choose the top K rows from the sorted array.
* Now, it will assign a class to the test point based on most frequent class of these rows.

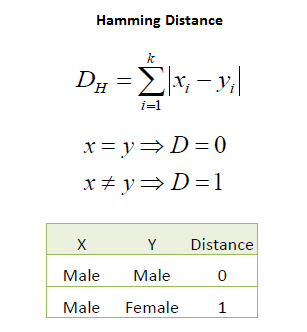
1. End

A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function. If K=1, then the case is simply assigned to the class of its nearest neighbor.



Equation 1: KNN Distance formula

It should also be noted that all three distance measures are only valid for continuous variables. In the instance of categorical variables the Hamming distance must be used. It also brings up the issue of standardization of the numerical variables between 0 and 1 when there is a mixture of numerical and categorical variables in the dataset.



Equation 2: Hamming Distance

Choosing the optimal value for K is best done by first inspecting the data. In general, a large K value is more precise as it reduces the overall noise but there is no guarantee. Cross-validation is another way to retrospectively determine a good K value by using an independent dataset to validate the K value. Historically, the optimal K for most datasets has been between 3-10. That produces much better results than 1NN.

**6.2 Naïve Bayes Algorithm**

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of [feature](https://en.wikipedia.org/wiki/Feature_vector) values, where the class labels are drawn from some finite set. There is not a single [algorithm](https://en.wikipedia.org/wiki/Algorithm) for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is [independent](https://en.wikipedia.org/wiki/Independence_(probability_theory)) of the value of any other feature, given the class variable. An advantage of naive Bayes is that it only requires a small number of training data to estimate the parameters necessary for classification.

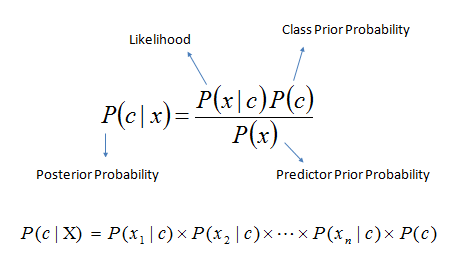
Bayes theorem provides a way of calculating the posterior probability, *P*(*c|x*), from *P*(*c*), *P*(*x*), and *P*(*x|c*). Naive Bayes classifier assume that the effect of the value of a predictor (*x*) on a given class (*c*) is independent of the values of other predictors. This assumption is called class conditional independence.

Algorithm:

1. Calculate the prior probability for given class labels
2. Find Likelihood probability with each attribute for each class
3. Put these value in Bayes Formula and calculate posterior probability.
4. See which class has a higher probability, given the input belongs to the higher probability class.

Characteristics of Naïve Bayes:

* They are robust to isolated noise points because such points are averaged out when estimating conditional probabilities from data.
* Naive Bayes classifiers can also handle missing values by ignoring the example during model building and classification.
* They are robust to irrelevant attributes.
* Correlated attributes can degrade the performance of naive Bayes classifiers because the conditional independence assumption no longer holds for such attributes.



Equation 3: Naïve Bayes

* *P* (*c|x*) is the posterior probability of *class* (*target*) given *predictor* (*attribute*).
* *P*(*c*) is the prior probability of *class*.
* *P* (*x|c*) is the likelihood which is the probability of *predictor* given *class*.
* *P*(*x*) is the prior probability of *predictor*.

Gaussian naïve Bayes:

In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a **Gaussian distribution**. A Gaussian distribution is also called [Normal distribution](https://en.wikipedia.org/wiki/Normal_distribution). When plotted, it gives a bell shaped curve which is symmetric about the mean of the feature values as shown below:



The likelihood of the features is assumed to be Gaussian, hence, conditional probability is given by:



Equation 4: Gaussian conditional probability

**6.3 Summary**

Naïve Bayes model is tremendously appealing because of its simplicity, elegance and robustness. It is one of the oldest formal classification algorithms, and yet even in its simplest form it is often surprisingly effective. Naive Bayes algorithms are mostly used in sentiment analysis, spam filtering and recommendation systems etc. They are fast and easy to implement. In most of the real life cases, the predictors are dependent, this hinders the performance of the classifier.

* 1. **Web application**

1. The users have to select the Symptoms from the dropdown as input. Based on symptoms selected the disease is predicted in the form of text.
2. The users will select the previous predicted disease from the dropdown as input, protein responsible for given disease is predicted as output in the form of text.
3. Then the user will select the previous predicted disease as input, treatment for given disease is predicted as output in the form of text.
   1. **Challenges Faced**
4. Data collection for symptoms, proteins and treatments
5. Converting data to train compatible to the code written
6. Inconsistent values for KNN which wasn’t meeting our functional requirements.
7. Embedding python code and dataset and making user interface friendly.
   1. **Challenges overcome**
8. Data collection was a major challenge so we have overcome by preparing the dataset manually. Initially we took protein-protein interaction and Disease gene association network dataset. By considering disease gene dataset, manually symptoms-disease, disease-protein, protein-disease and disease-treatment datasets are generated.
9. The problem in KNN algorithm was solved by applying naïve Bayes algorithm.
10. Flask framework has used for embedding python code and HTML, CSS code to make User Interface friendlier.
    1. **Applications**

* **Health care**

The model is useful by doctors to find out the proteins responsible for the disease and come up with a root analysis of the disease and come up with treatment for the root of the problem than symptomatic treatment.

**CHAPTER-7**

**TESTING**

Software Testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding.

The main objective of testing is:

* Executing a program with intent of finding an error.
* A good test case is one that has a high probability of finding a yet undiscovered error

Testing is done on Machine Learning side using Test and Train data sets. Various combinations of Data Sets are formed and degree accuracy is calculated to ensure that algorithm has lesser deviations. Machine Learning will identify if the algorithms give right degree of accuracy using prediction validators.

Large systems are built out of sub-systems, which are built out of modules, which are composed of procedures and functions. Testing process therefore proceed in stages where testing is carried out incrementally in conjunction with system implementation.

* **The test cases designed for the projects are discussed below:**

|  |
| --- |
| Test case 1: Submit the diseases name from the list |
| Preconditions: The applications is open. |
| Assumptions: The protein id for the diseases are available |
| Test steps: 1. Select the checkbox from the list  2.Click result |
| Expected Result: The disease selected should be submitted and protein id with high probability is displayed as result. |

**Fig 7.1: Testcase 1**

|  |
| --- |
| Test case 2: Submit the diseases name from the list |
| Preconditions: The applications is open. |
| Assumptions: The treatments for the diseases are available |
| Test steps: 1. Select the checkbox from the list  2.Click result |
| Expected Result: The disease selected should be submitted and treatment for the diseases is displayed as result. |

**Fig 7.2: Testcase 2**

|  |
| --- |
| Test case 3: Select multiple protein id ‘s from the list |
| Preconditions: The applications is open. |
| Assumptions: The diseases for the protein ids are available |
| Test steps: 1. Select the checkbox from the list  2.Click result |
| Expected Result: The multiple protein id’s selected should be submitted and diseases name is displayed as result. |

**Fig 7.3: Testcase 3**

|  |
| --- |
| Test case 4: Select multiple Symptoms from the list |
| Preconditions: The applications is open. |
| Assumptions: The symptoms for the diseases are available |
| Test steps: 1. Select the checkbox from the list  2.Click result |
| Expected Result: The multiple symptoms selected should be submitted and diseases name is displayed as result. |

**Fig 7.3: Testcase4**

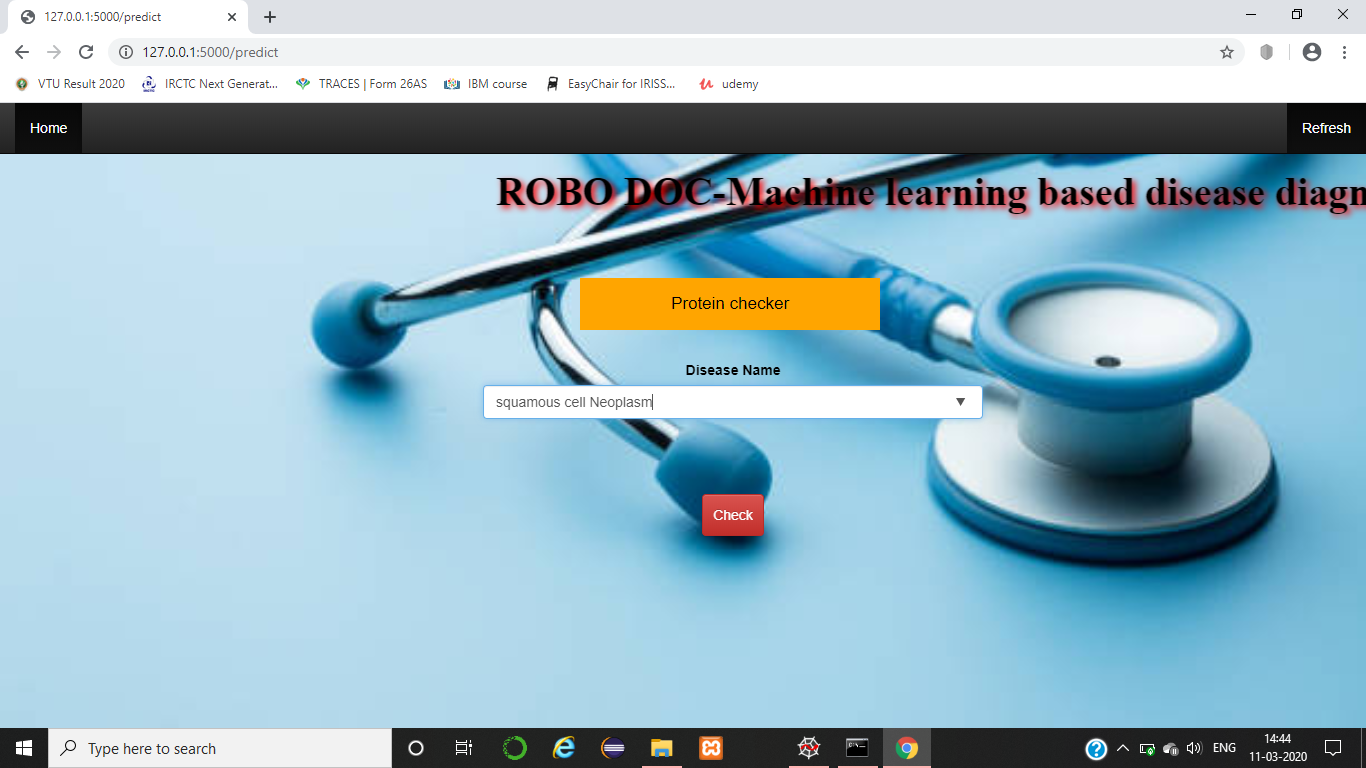
* Few testcases are explained above for each dataset, similarly all the cases work and outcomes are as expected.
* Further the accuracy of model is calculated for each dataset and below are percentages obtained.
* The model is tested for all the datasets and the accuracies are **100% for symptoms, 29% for treatment, Disease input – protein output 40% and Protein input – disease output 100% applying Naïve Bayes algorithm.**

**CHAPTER- 8**

**RESULT ANALYSIS**

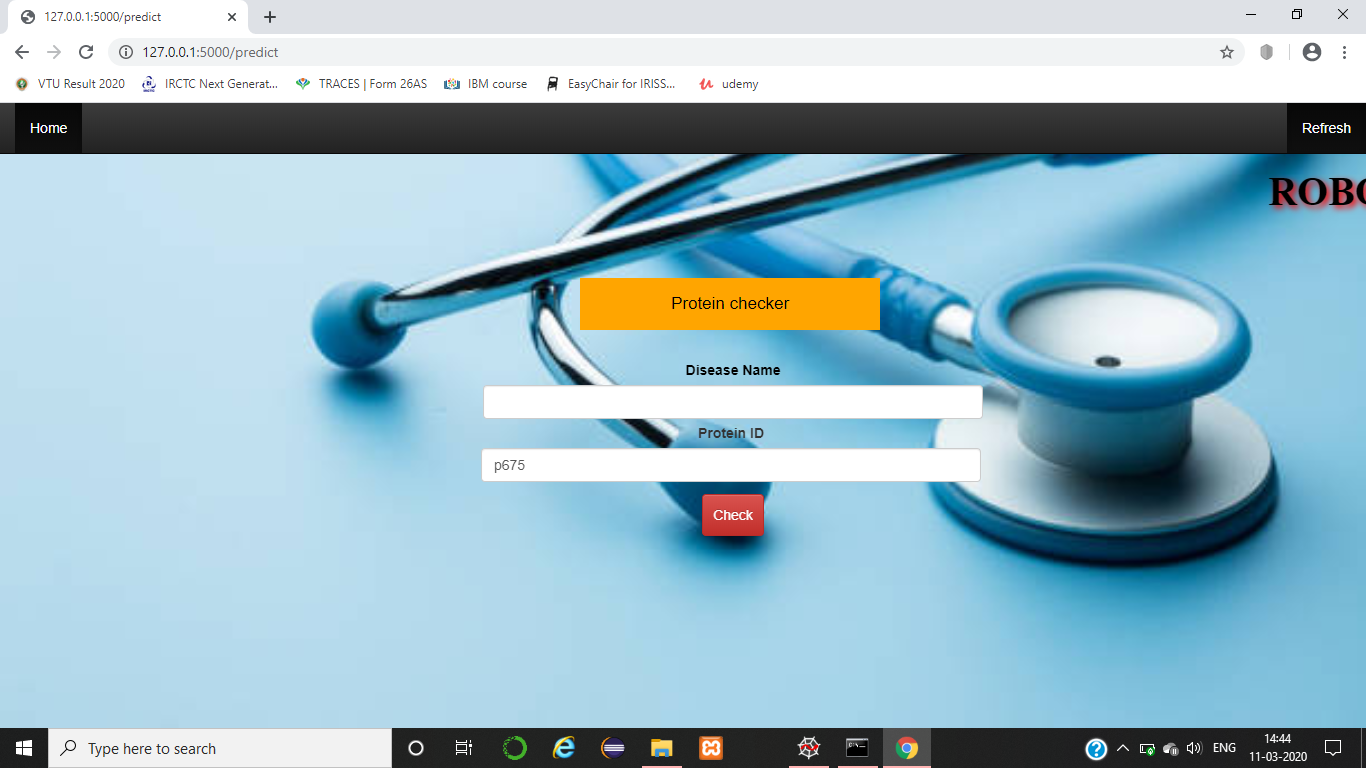
A result is the final consequence of actions or events expressed qualitatively or quantitatively. Performance analysis is an operational analysis, is a set of basic quantitative relationship between the performance quantities.

**8.1 Screenshots**

****

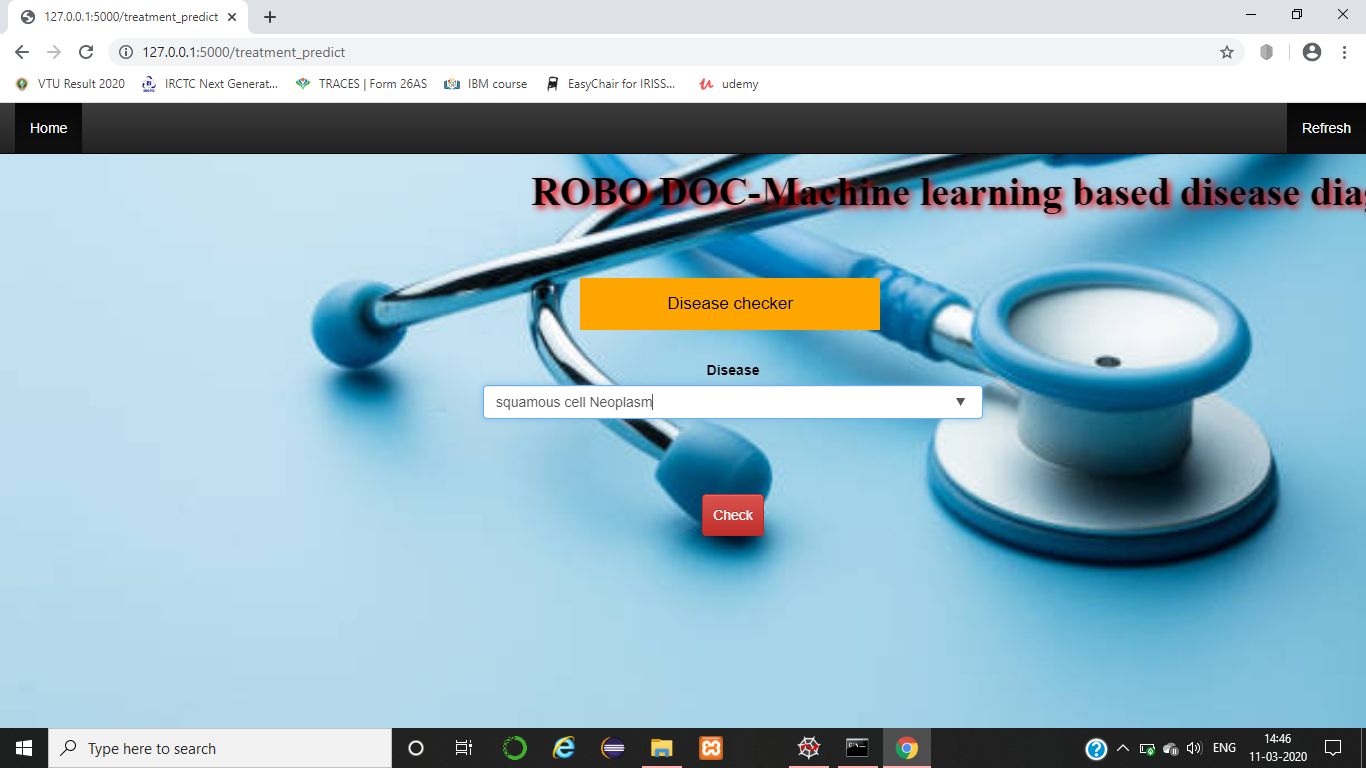
**Fig 8.1: Protein checker for disease**

The above snapshot represents the protein checker interface which takes disease name as input to display respective protein id.

****

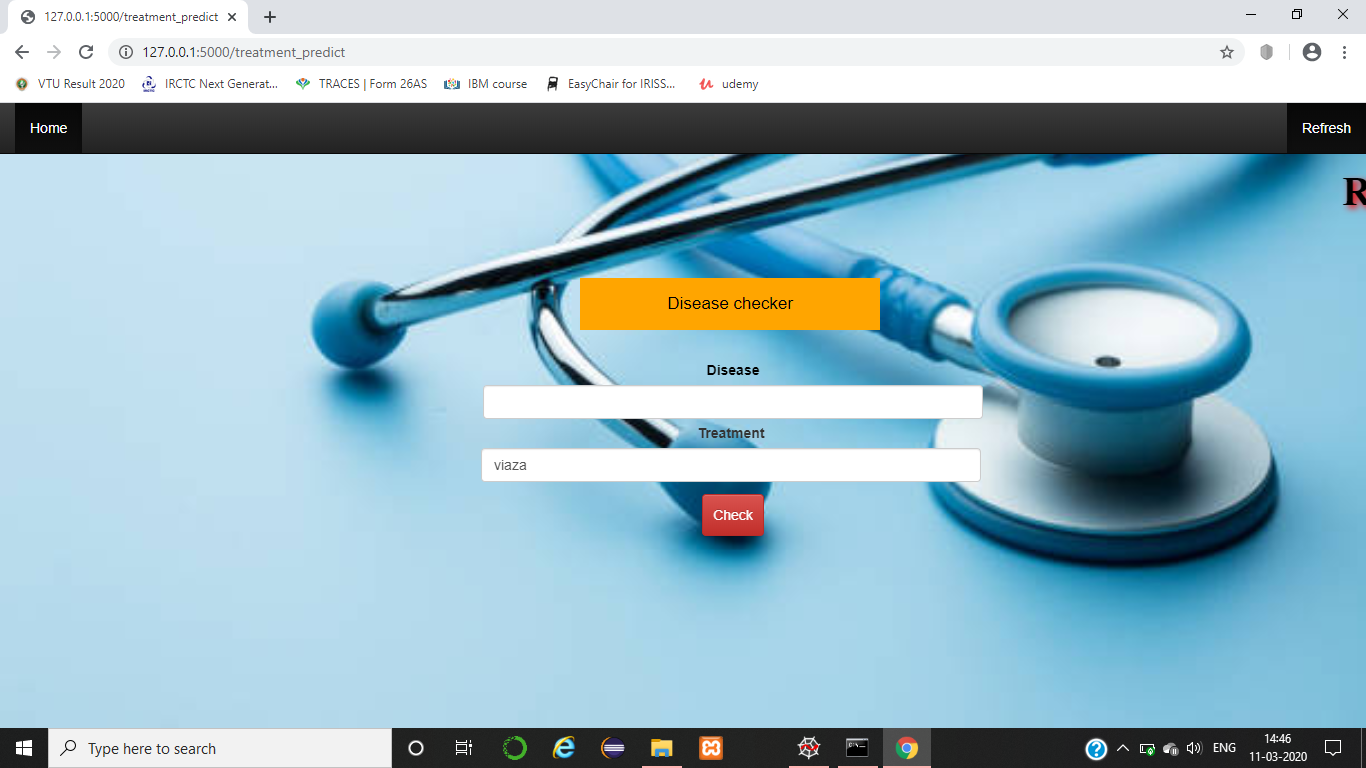
**Fig 8.2: Outcome of Protein checker for disease**

The above snapshot represents the protein checker interface where protein id is displayed for the input given.

****

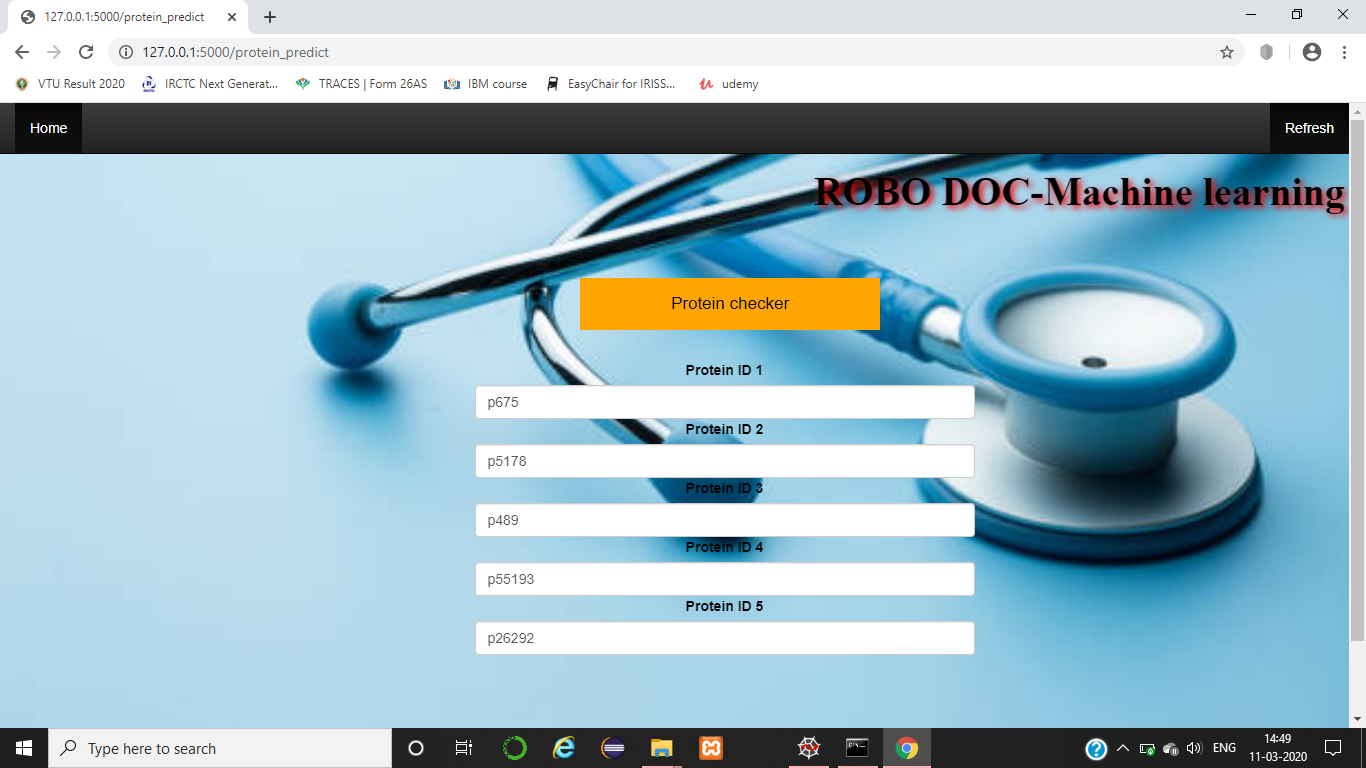
**Fig 8.3: Disease checker for treatment**

The above snapshot represents the Disease checker interface which takes disease name as input to display respective treatment for the disease.



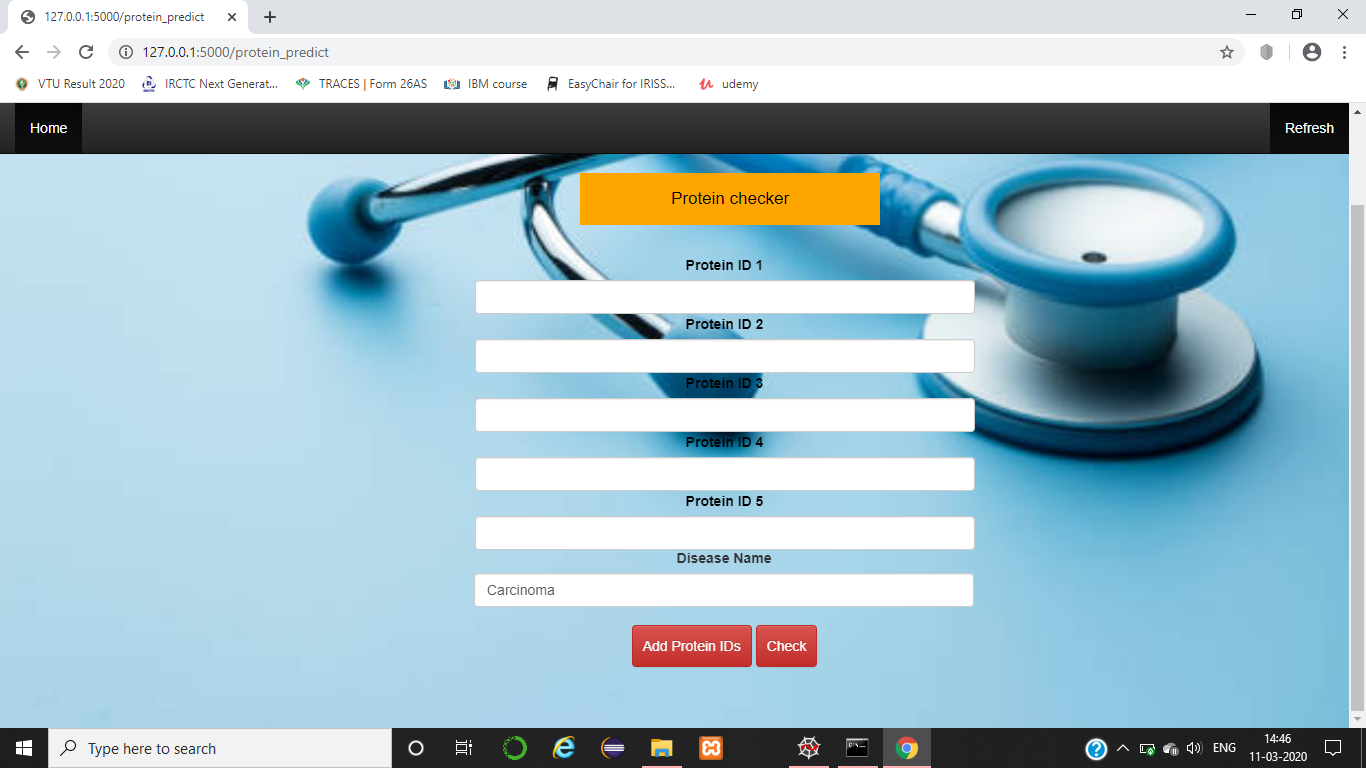
**Fig 8.4: Outcome of Disease checker for treatment**

The above snapshot represents the Disease checker interface where respective treatment is displayed for the input given.

****

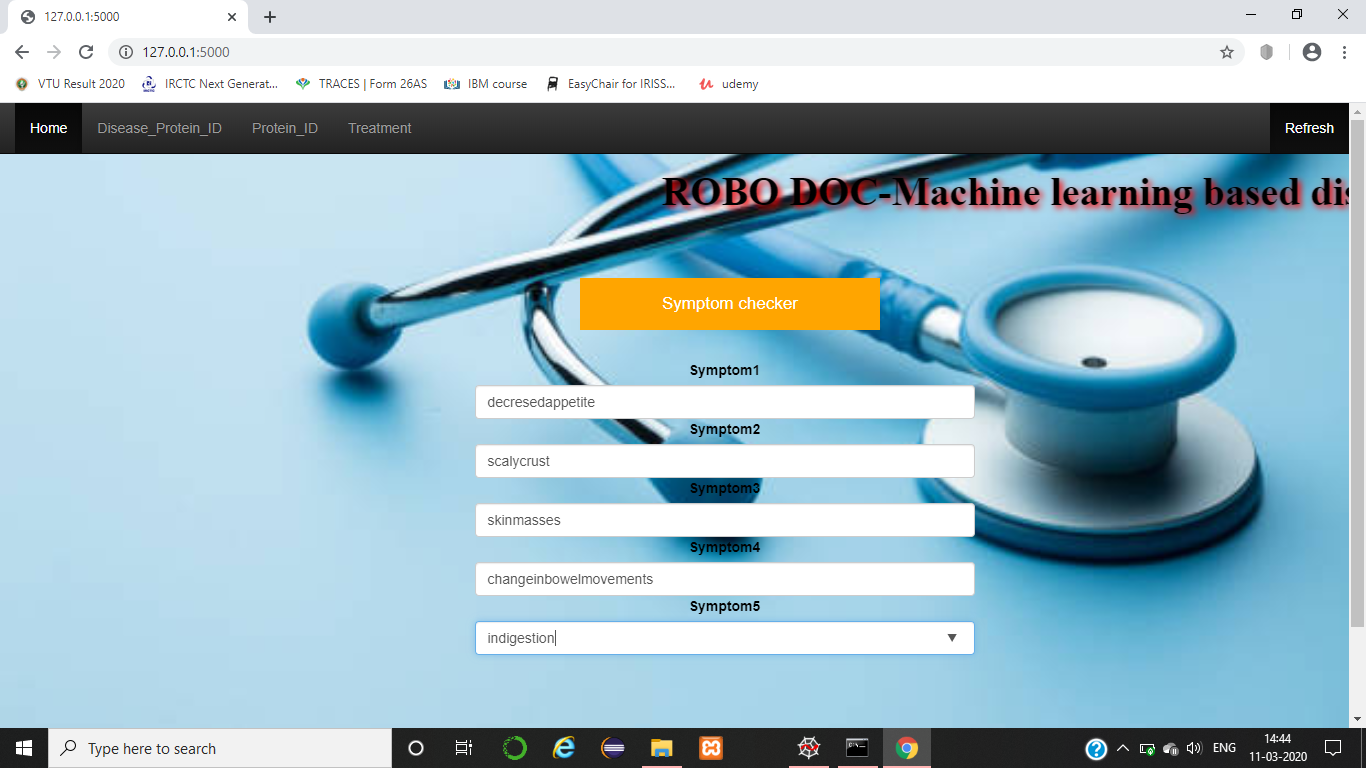
**Fig 8.5: Disease checker for protein id’s**

The above snapshot represents the Disease checker interface which takes multiple protein id’s as input to display respective diseases.



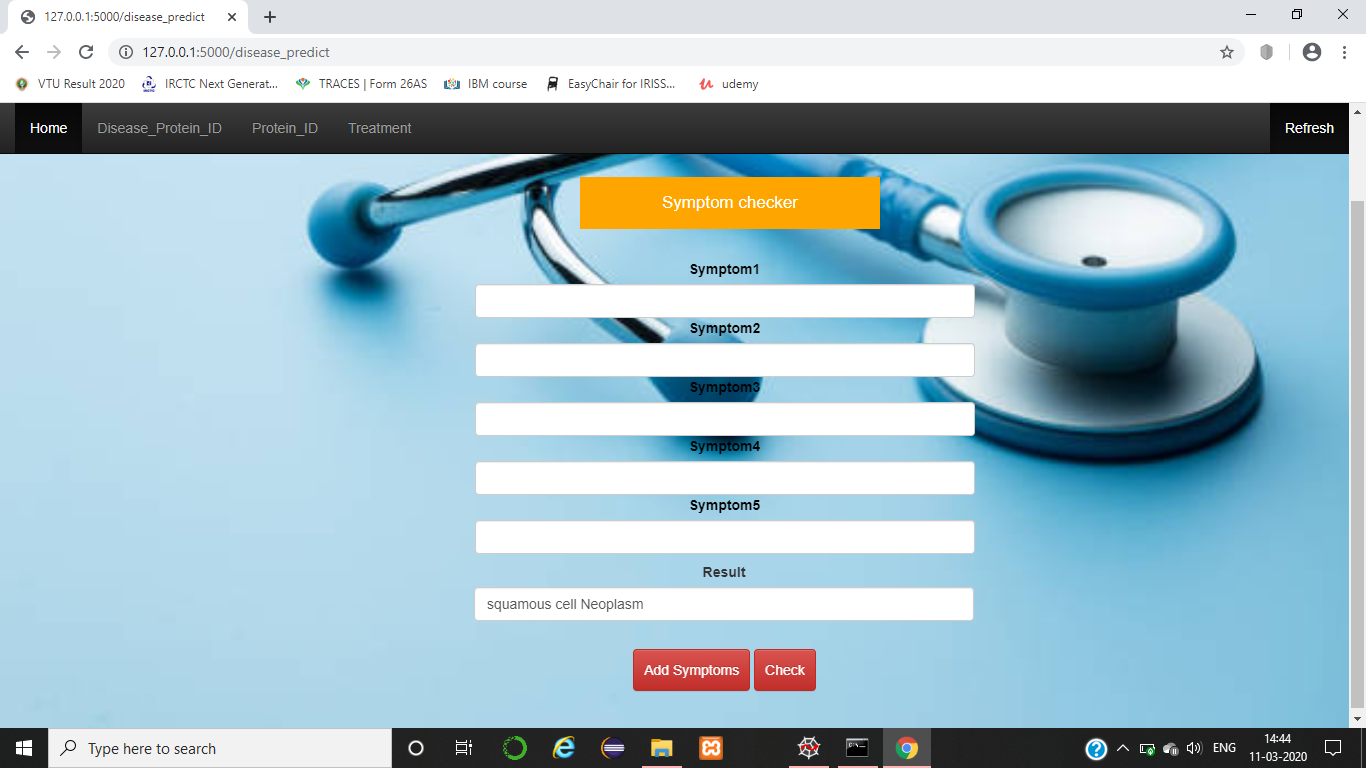
**Fig 8.6: Outcome of Disease checker for protein id’s**

The above snapshot represents the Disease checker interface where disease name is displayed for the list of protein id’s given as input.

****

**Fig 8.7: Symptom checker for diseases**

The above snapshot represents the Symptoms checker interface which takes multiple Symptoms as input to display respective diseases.



**Fig 8.8: Outcome of Symptom checker for diseases**

The above snapshot represents the Symptom checker interface where disease name is displayed for the list of symptoms given as input.

****

**Fig 8.9: protein - protein interaction**

**Fig 8.10: KNN for protein - protein interaction**

The above snapshot represents results of KNN classification algorithm yielded in 70% accuracy considering 2 neighbors and 400 proteins per disease

**Fig 8.11: Naïve Bayes Accuracy**

Th graph pits the accuracy obtain for different predictions using different inputs using Naïve Bays Algorithm.

**CHAPTER-9**

**CONCLUSION**

Based on symptoms, the proposed system is capable of classifying diseases and the protein- protein interactions that are responsible for diseases. The proposed system can suggest treatment based on symptoms and disease identified. That is, the possible treatments are identified and the best one would be suggested by the model based on its knowledge acquisition and rendition. The data repository makes itself a unique on as there is no such repository which gives all the information about symptoms, proteins responsible for the diseases and the best treatment possible for genetic disease the model works on. The built model is useful by doctors to find out the proteins responsible for the disease and come up with a root analysis of the disease an come up with treatment for the root of the problem than symptomatic treatment.

**9.1 Future work**

Apart from invasive procedures, diseases can be diagnosed using machine learning aid with better accuracy compare to previous work.

Further work includes applying reverse engineering which can predict protein interactions considering only symptoms and propose possible treatment.

**CHAPTER-10**

**REFERENCES**

1. <https://web.stanford.edu/class/cs273a/cgi-bin/>
2. <https://www.frontiersin.org/articles/10.3389/fgene.2019.00535/full>
3. <https://cs.stanford.edu/people/jure/pubs/ohmnet-ismb17.pdf>
4. h[ttps://scikitlearn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html)
5. <https://www.tutorialspoint.com/flask/index.htm>
6. <https://pythonspot.com/flask-web-app-with-python/>
7. h[ttps://www.biorxiv.org/content/biorxiv/early/2019/01/28/532226.full.pdf](https://www.biorxiv.org/content/biorxiv/early/2019/01/28/532226.full.pdf)
8. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3671239/>
9. <https://string-db.org/cgi/input.pl?sessionId=ohBWzrJORQR1&input_page_show_search=on>
10. <https://string-db.org/cgi/input.pl?sessionId=ohBWzrJORQR1&input_page_active_form=single_identifier>
11. <https://string-db.org/cgi/input.pl?sessionId=ohBWzrJORQR1&input_page_active_form=multiple_sequences>
12. Ravindhar NV, Anand, Hariharan Shanmugasundaram, Ragavendran, Godfrey Winster, Intelligent Diagnosis of Cardiac Disease Prediction using Machine Learning, September 2019, 8, 11.
13. Anooja Ali, Vishwanath R, Hulipalled, S. S. Patil, Raees Abdulkader, Alignment of Protein Interaction Networks and Disease Prediction: A Survey, August 2019,8,4.
14. Pratik Dutta, Sriparna Saha, Saurabh Gulati, Graph-Based Hub Gene Selection Technique Using Protein Interaction Information: Application to Sample Classification, January 2019, 23,6.
15. Satyabrata Aich, Hee-Cheol Kim, Kim younga, Kueh Lee Hui, Ahmed Abdulhakim Al-Absi, Mangal Sain, A Supervised Machine Learning Approach using Different Feature Selection Techniques on Voice Datasets for Prediction of Parkinson’s Disease, May 2018, 7,3.
16. Hina Umbrin, Saba Latif, A Survey on Protein Protein Interactions, methods, databases, challenges, and future directions ,2018.
17. Pahulpreet Singh Kohli, Shriya Arora, Application of Machine Learning in Disease Prediction, 2018.
18. Tanlin Sun, Sequence-based prediction of protein protein interaction using a deep-learning algorithm,vol. 18, no. 1, pp. 277, 2017.
19. Murakami Yoichi, Network analysis and in silico prediction of protein-protein interactions with applications in drug discovery, vol. 44, pp. 134-142, 2017.
20. Md. Tahmid Rahman Laskar, Md. Tahmid Hossain, Abu Raihan Mostofa Kamal, Nafiul Rashid, Automated Disease Prediction System (ADPS): A User Input-based Reliable Architecture for Disease Prediction, January 2016,133,15.
21. Antanaviciute,A. et al, GeneTIER: prioritization of candidate disease genes using tissue-specific gene expression profiles. Bioinformatics, 2015,31, 2728–2735.
22. P.Creixelletal.,NatureMethods12,p.615(2015).
23. J.Piñeroetal.,Database2015(2015).
24. M. D. Ritchie, E. R. Holzinger, R. Li, S. A. Pendergrass and D. Kim, Nature Reviews Genetics 16, 85 (2015).
25. Ganegoda,G.U. et al. (2014) Prediction of disease genes using tissue-specified gene-gene network. BMC Syst. Biol., 8, S3.Li, Y. and Li, J.
26. Fagerberg,L. et al, Analysis of the human tissue-specific expression by genome-wide integration of transcriptomics and antibody-based proteomics. Mol. Cell. Proteom., 2014,13, 397–406.
27. M.Gustafssonetal.,GenomeMedicine6,p.82(2014).
28. Jacob Köhler, Graph-based analysis and visualization of experimental results with ONDEX, vol. 22, no. 11, pp. 1383-1390, 2006.
29. Barutcuoglu,Z. et al. Hierarchical multi-label prediction of gene function. Bioinformatics, 2006, 22, 830–836.
30. Ashburner,M. et al., Gene Ontology: tool for the unification of biology. Nat. Genet.,2000, 25, 25–29.