B. TECH. PROJECT REPORT

On

Design and Development of Medical Diagnosis Expert System using One Class Classifiers along with Statistical Methods

BY

Ketan Uday Chaware 13000 1009

Vishwajeet Singh Thakur 13000 1040

DISCIPLINE OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

NOVEMBER 2016

Design and Development of Medical Diagnosis Expert System using One Class Classifiers along with Statistical Methods

A PROJECT REPORT

Submitted in partial fulfillment of the
requirements for the award of the degrees

of
BACHELOR OF TECHNOLOGY
in

COMPUTER SCIENCE AND ENGINEERING

Submitted by:
Ketan Uday Chaware 13000 1009
Vishwajeet Singh Thakur 13000 1040

Guided by:
Dr. Aruna Tiwari,
Assistant Professor

INDIAN INSTITUTE OF TECHNOLOGY INDORE
November 2016

CANDIDATE'S DECLARATION

We hereby declare that the project entitled **"Design and Development of Medical Diagnosis Expert System using One Class Classifiers along with Statistical Methods**" submitted in partial fulfillment for the award of the degree of Bachelor of Technology in 'Computer Science and Engineering' completed under the supervision of **Dr. Aruna Tiwari, Assistant Professor, IIT Indore** is an authentic work.

Further, we declare that we have not submitted this work for the award of any other degree elsewhere.

Ketan Uday Chaware **Vishwajeet Singh Thakur**

CERTIFICATE by BTP Guide

It is certified that the above statement made by the students is correct to the best of my knowledge.

Dr. Aruna Tiwari,
Assistant Professor, CSE,
Indian Institute of Technology Indore

Preface

This report on "Design and Development of Medical Diagnosis Expert System using One Class Classifiers along with Statistical Methods" is prepared under the guidance of Dr. Aruna Tiwari. This report is submitted in partial fulfillment of requirements for award of degree of Bachelor of Technology in Computer Science and Engineering at Indian Institute of Technology Indore.

In this report we propose new model for Auto-Encoder i.e. neural network used for One Class Classification which tries to overcome the pitfalls of current models which are unable to handle dynamism. Our model utilizes best from both – neural learning models and statistical algorithms to tackle the dynamism and hence is a hybrid model. We developed this model for problem of automating process of medical diagnosis which has inherent dynamism associated with it.

This report thoroughly explains all aspects of new model, its design, its implementation, experimentations, results and conclusions and finally, it gives insights on its usability and scope for future work and development. Moreover, this report is written in top-down fashion to facilitate the understanding i.e. we start with overview, then high level design and then go into details of each component.

We thank everyone for their direct or indirect support in making this report a success. With this, we conclude the preface.

Ketan Uday Chaware
Vishwajeet Singh Thakur

B.Tech. IV Year
Discipline of Computer Science and Engineering
IIT Indore

Acknowledgements

We wish to thank Dr. Aruna Tiwari for her kind support and valuable guidance. It is her help and support, due to which we became able to complete the design and technical report. Without their support this report would not have been possible.

We thank Mr. Chandan Gautam for his help and support at every step in completing the project. We will like to remain in debt of Dr. Bude and Dr. Khatree who provided us valuable guidance and helped in data collection in spite of their busy schedule. We also thank our parents, family members and friends for their unconditional support. Finally we would like to thank all those who helped us knowingly or unknowingly.

Ketan Uday Chaware
Vishwajeet Singh Thakur

B.Tech. IV Year
Discipline of Computer Science and Engineering
IIT Indore

Abstract

Neural networks are widely used in many areas today for various problems. But they lack the ability to handle dynamic input and output vectors and thus restricting their applicability to many domains which have inherent dynamism e.g. medical diagnosis. The project aimed at modifying Auto-Encoder (feed-forward neural network used in One Class Classification) and enabling it to handle dynamism. The modified model is able to handle dynamic and varying sized input as well as output vectors. This was achieved, partially by modifying network model and partially by blending it with statistical methods. A framework was developed for handling dynamism in auto-encoders which in future can be extended to other neural networks. Developed model was then tested by applying it to problem of medical diagnosis and performing experimentations. The accuracy levels obtained are good enough for making model deployable for this problem. This project is a small step towards making neural networks capable of handling dynamism and hence expanding their applicability to large number of problems where dynamism restricts their ability to those domains.

Table of Contents

Chapter 1

Introduction

This chapter is intended to introduce the reader to the problem at hand. Chapter clearly states objectives and highlights roadblocks faced in achieving them. It also provides insights on why this project is required. Finally it gives the overall outline of remaining thesis.

Field of medical diagnosis is closely related to our life. Process of medical diagnosis i.e. detecting the disease given the symptoms faced by the patient, is foremost step of any treatment. It is one of the basic needs of mankind. And still in spite of all the technological advancements, we are underperforming in this field. In developing countries like India, there is acute shortage of good medical practitioners and especially in rural parts situation is pathetic. Even in developed countries like US, cost of diagnosis is very high and out of reach of poor communities. Increasing cases of medical mal-practices like prescribing costly medicines and unnecessary test, performing unnecessary surgeries etc. are evident now-a-days. One of the possible reason for this situation is that even when automation flourishes in most other fields and industries, field of medical diagnosis lacks it significantly.

The process of automating diagnosis of disease is extremely challenging. This field is very dynamic with discoveries of new diseases every year. Each disease can have different symptoms associated with it in different patients. Also there is no mathematical model or logic in disease-symptom relationship. Many times it even lacks biological model, leave alone the mathematical model. High accuracy requirements makes it even more difficult as lives of patients are at stake. Also hidden factors like medical history, drug allergies, epidemics, weather etc. add to the complexity of already complex problem.

Our project takes a small step in direction of automating the process of medical diagnosis and hence expanding its reach to common man. In this project we developed an application to assist medical practitioners in process of diagnosis and treatment. We tried to overcome few of the roadblocks mentioned above. We achieved this by blending machine learning approach with statistical methods to tackle the dynamism and lack of mathematical model. Requirement of high accuracy can be handled by training with large number of examples and so we implemented our application to be able to work on distributed platform to get large number of simultaneous users. Modelling hidden factors remains as future scope.

While developing this application and dealing with all these roadblocks, we developed a model which is able to learn from training data and simultaneously able to handle situation of dynamic input and output vector sizes. All of the current learning algorithms lack this ability right now. In this project we tried and partially succeeded in incorporating this capability into neural networks.

Now we will provide an overview of developed system followed by analysis of problem and literature survey. Then we provide detailed explanation of design and architecture. After that we give details of implementation and experimental results. We will conclude with benefits and future scope.

Chapter 2

Overview

This chapter gives overview of developed application. It explains in details everything about what the developed application is, what it does and what is use of it.

As said earlier in introduction, we developed a tool to assist medical practitioners in process of diagnosis and treatment. A point to note here is that the tool will only assist doctors and nor replace them. Now let's have a tour of developed application. There are five steps for diagnosis and treatment and we explain each of them in details in this section.

2.1 Steps in diagnosis

Step 1: Collecting general information.

In first step application collect general information about patient like age, gender, blood pressure, sugar levels etc. Screenshot can be seen below:

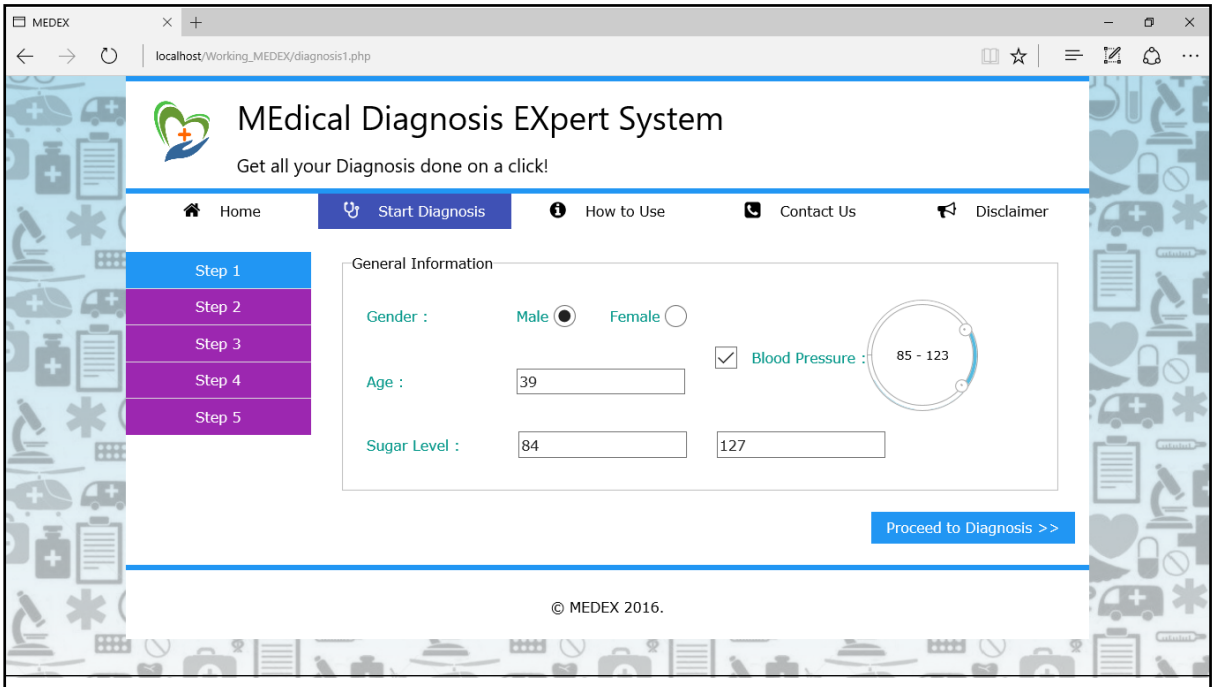


Figure 2.1: *Step 1- Collection of general information about patient.*

Step 2: Collecting information about symptoms.

In second step, we collect information about symptoms from which patient is suffering. Each symptom is associated with intensity value e.g. on scale of 1 to 5 how high is your head ache? Also there are two types of symptoms – one which involves body parts e.g. head ache and other which doesn't involve body part e.g. fever. Below is the screenshot:

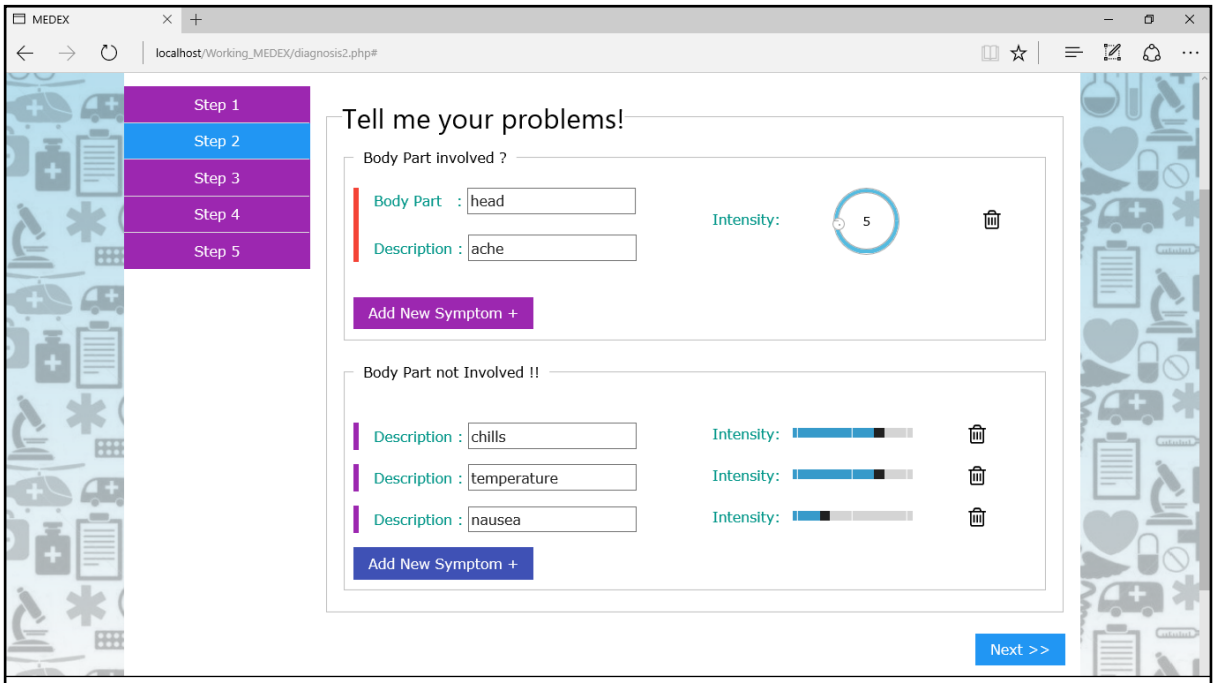


Figure 2.2: *Step 2 - Collection information about symptoms.*

Step 3: Asking about missing symptoms.

In this step we ask user about possibly missing symptoms. E.g. suppose user inputs symptoms that gives application 'the feeling' that patient may be suffering from malaria. But suppose user haven't inputted "head ache" which is important symptom for malaria. We call it a "missing symptom". So now application asks user if patient is indeed having head ache. This information is used to increase confidence of application about that particular diagnosis. Screenshot can be seen on next page.

Step 4: Floating out list of all possible diagnoses.

Now we provide user i.e. doctor with the list of all possible diagnoses along with confidence factors – the value that shows confidence of application about that diagnosis. User will now have to choose one of the diagnosis which he thinks is correct to get latest treatment options for that disease. If user thinks that none of the diagnosis from list is correct, he can provide his own input using 'Other' option (seen in screenshot on next page). This is very crucial step in application as this is where it learns from the doctor about correct diagnosis. It updates it knowledge repository and trains its machine learning components based on input from doctor. So it basically gains the expertise from doctor every time system is used. Screenshot on next page.

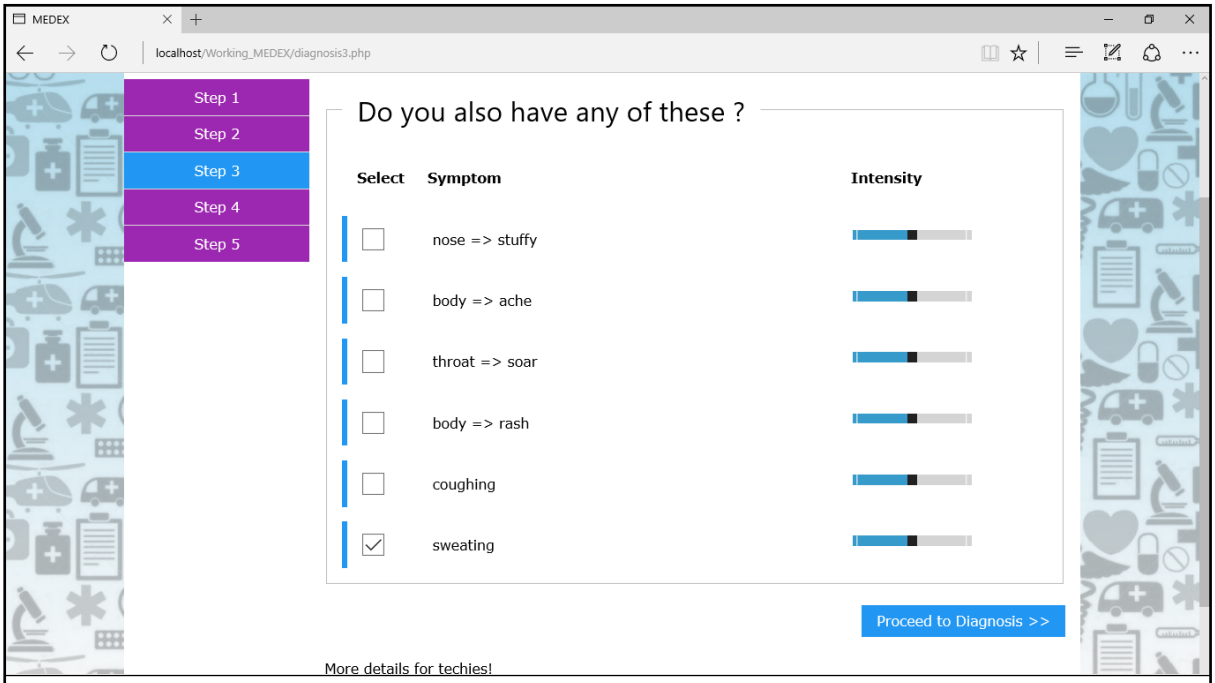


Figure 2.3: *Step 3 - Asking about missing symptoms.*

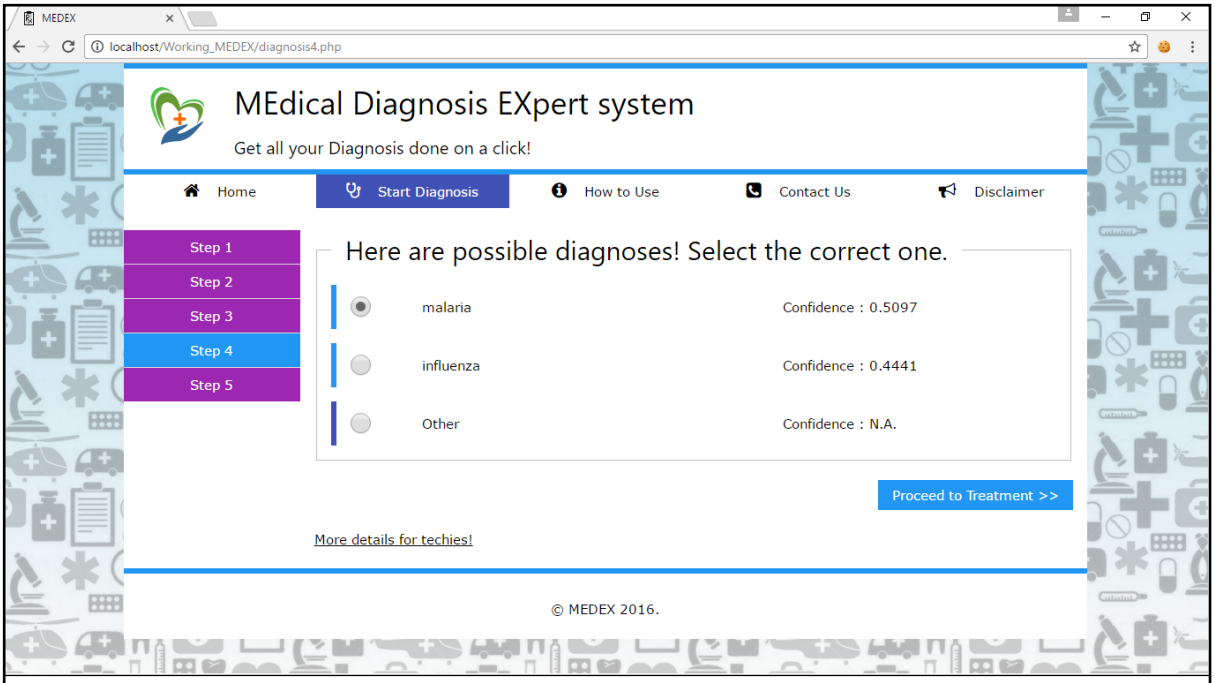


Figure 2.1: *Step 4 - Floating list of all possible diagnoses.*

Step 5: Provide the latest treatment options.

This is the last step in the process where application provides doctor with latest available generic treatment (prescribing drugs instead of medicines) options for selected diagnosis. So doctor can brush up their memory and prescribe medicines to patient accordingly. Screenshot is given below:

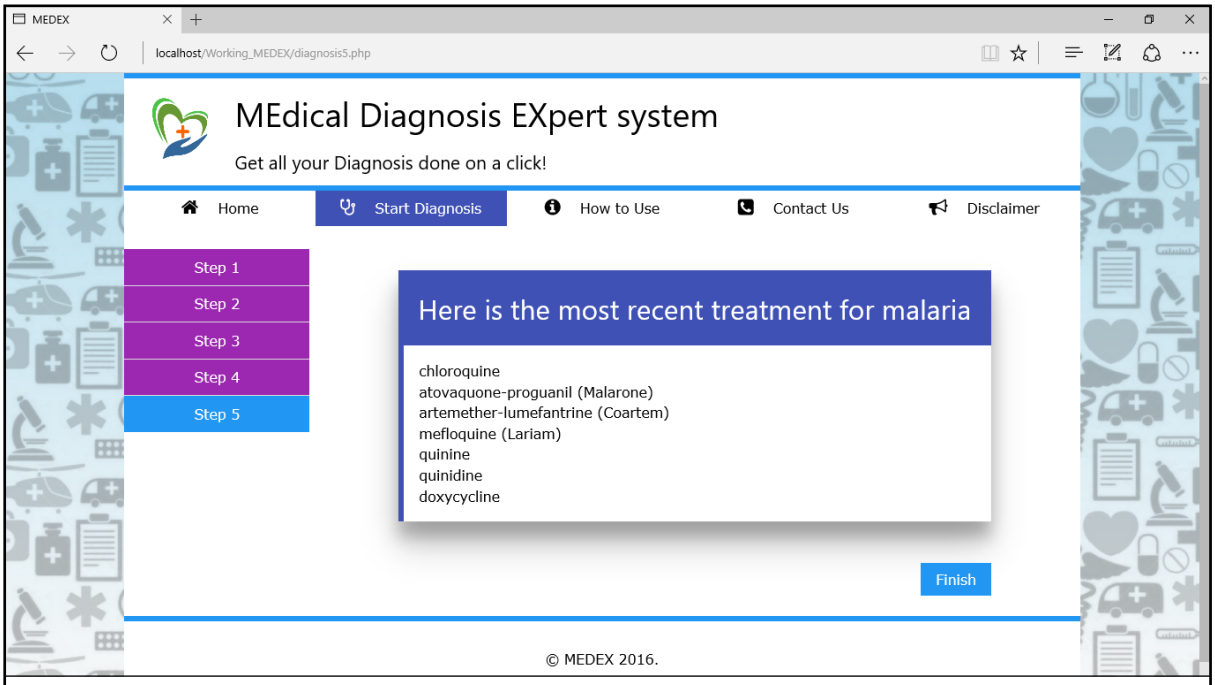


Figure 2.1: *Step 5 - Providing latest treatment options.*

2.2 Benefits to doctors:

1. **Never miss out any important symptom** – Even if patient forgets to mention about some important symptom, application will ask user about that in third step and so now we never miss out important symptom.
2. **Never miss out any possibility** – In fourth step, user gets list of all possible diagnoses and so they will never miss out any possible option just because they don't have perfect memory like computers.
3. **Get the latest treatment options** – In last step, user gets latest treatment option for selected disease. Now he can refresh his memory and prescribe medicines based on options provided by system. After all doctors don't have enough time to keep up with latest treatment of all diseases.

Chapter 3

Analysis of Problem and Literature Survey

In this chapter a thorough analysis of problem statement is provided. It explains and justifies the choice of technology for various components. It also provides literature survey of existing technology, problems with it and modification that we need to do and hence actually builds grounds for understanding architecture of application.

In last chapter, we have got insights of application and its utility. In this chapter, we will do a thorough analysis of problem and decide which technologies to use for each component in the application.

3.1 Analysis of problem and Choice of Technology

As stated earlier field of medical diagnosis is very dynamic and fixed mathematical models are not at all suitable for this field (and actually they cease to exist!). So machine learning (ML) paradigm which can learn the concepts from seemingly unstructured data and can adopt themselves to changing environment was a necessary component to be included. But almost all ML techniques fail to handle dynamic size of input and output vectors which is inherent to process of diagnosis. So ML alone would have failed miserably to handle such a dynamic field. So we decided to include statistical methods to support ML tackle the dynamism at input and output level.

In diagnosis, we need to identify disease from given set of input symptoms i.e. we need to attach a disease label to each given set of symptoms which means we need to classify given set of symptoms into possible diseases. So from huge ML paradigm, we chose classification based approaches. Now there are two types of classifiers – One class and Multi Class (This is one of taxonomies of many available). In One Class Classifiers (OCC) there is single output class and classifier tells whether or not given input belongs to that class. While in Multi Class Classifiers (MCC) we have several output classes and system tells to which class the given input belongs. Consider MCC first. We can have separate class for every disease and train the system. Every time user inputs the symptoms, we fed it to the MCC and get to know which disease patient may be suffering from. But there is major flaw in logic. Suppose system discovers a new disease. It can't add new class without re-training whole system. That means all the previous training goes in vein. This makes system very unstable and renders it useless. With OCC, there is separate classifier for each disease and so addition of new disease will not affect any other OCCs at all. So obviously OCC was the way to go ahead with.

For statistical models, almost all existing models require assumptions about probabilistic distribution of data. But we don't have any mathematical model in data. So we decided to develop our own statistical algorithm to tackle the problem and support the ML component in its task. This is how we finalized the technology stack.

3.2 Literature Survey of OCC

Now that we have decided to use OCC as one of the most important component of architecture, let's have look at various types of OCCs and their applicability to our problem.

3.2.1 Basics of OCC

As we said earlier, OCCs answer the question "does object belongs to a particular class or not?" instead of answering "to which class does object belong?" We use OCC when we have sufficient training sample belonging to one class (which we refer as positive or target class arbitrarily) and the other class (negative or outlier) is significant under-sampled or un-sampled. So only one side of classification boundary can be obtained using positive data. This makes dealing with OCCs harder compared to Multi-class classifiers. A special care has to be taken to simulate error due to outliers because they are absent. Otherwise classifiers will go on accepting everything instead of learning features from data. In next section, we study different types of OCCs and chose the suitable one for our case.

3.2.2 Taxonomy for OCC

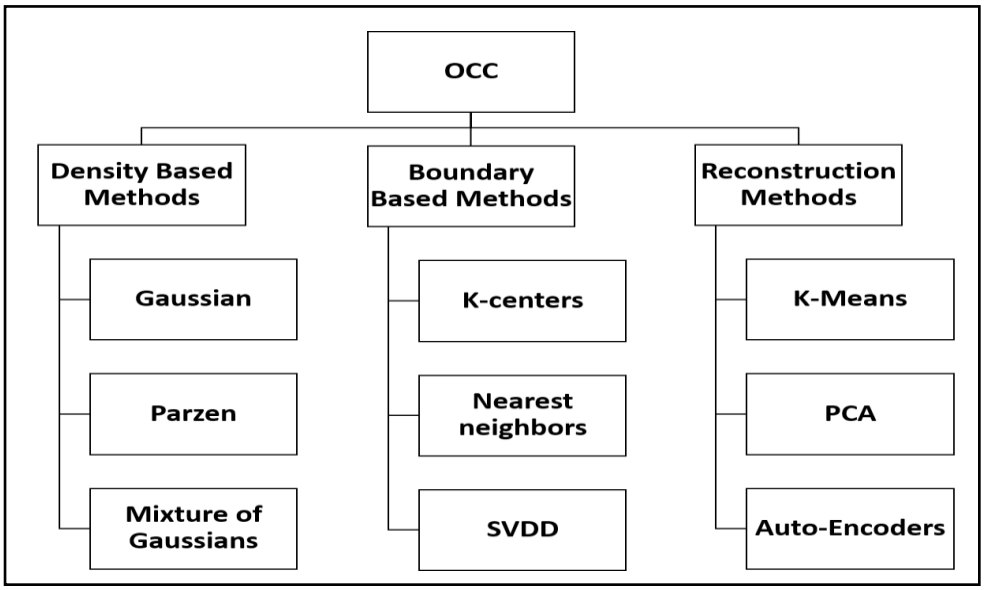


Figure 3.1 : *Taxonomy of OCC*

There are various taxonomies proposed for study of One Class Classifiers. Each uses different characteristics of OCC to classify them. Given figure is simplified version with only few examples from each class of taxonomy proposed by Mazhelis, O [6]. It classifies OCCs based on internal model used by them. (This is a simplified version. For detailed taxonomy, refer the original paper.) For other taxonomies please refer Khan, S.S. and Madden, M.G. [8] and Tax, D.M.J [1]. Now we explain in short about each type of OCC.

Density based methods

As name suggests, this method is based on estimation of probability distribution of feature values in whole feature space. This distribution is then used to calculate errors due to outliers as we can't do that directly due to absence of outliers in training set. Now we try to minimize classification error as well as error due to outliers to get the classifier boundary. This method is not suitable for our problem as we don't have any mathematical modelling of disease-symptom relationship and hence no probability distribution can be assumed.

Boundary based methods

In this methods, OCC tries to build a boundary around the data using the training data. At same time it tries to keep the enclosed volume as less as possible to avoid acceptance of outlier objects. This gives a tighter boundary around training data. Now distance of testing object from this boundary can be used to decide if test object belongs to given class or not. This techniques requires all training sets to be present at time of deciding boundary i.e. these methods are not suitable for streaming data. As capability to handle streaming data is key requirement in our problem, these methods are not very useful.

Reconstruction methods

In reconstruction methods, we try to reconstruct input at the output. First step is encoding the input and in next step we decode it. Now Euclidean distance between decoded output and original input gives accuracy of reconstruction which is measure of membership of test object to target class. While encoding and decoding, it is important to bottleneck the flow of test data so that model learns about concept instead of memorizing it. This method looks suitable for our problem as it don't require any assumption about mathematical models and hence gives more flexibility in design. But again K-means and PCA are not very suitable. K-means is difficult to modify for online sequential case and some parameters in that methods are difficult to figure out in case of medical data. PCA though is more suitable for high dimensional data, is very difficult to modify for variable dimensional data as it deals with mapping set of variables into smaller dimensional space still preserving most of variance of data. Auto-encoders on other hand are neural networks which are easy to modify for dynamic input size as well as streaming data. So auto-encoders was chosen as technique to go ahead with.

3.2.3 Basics of Auto-Encoders

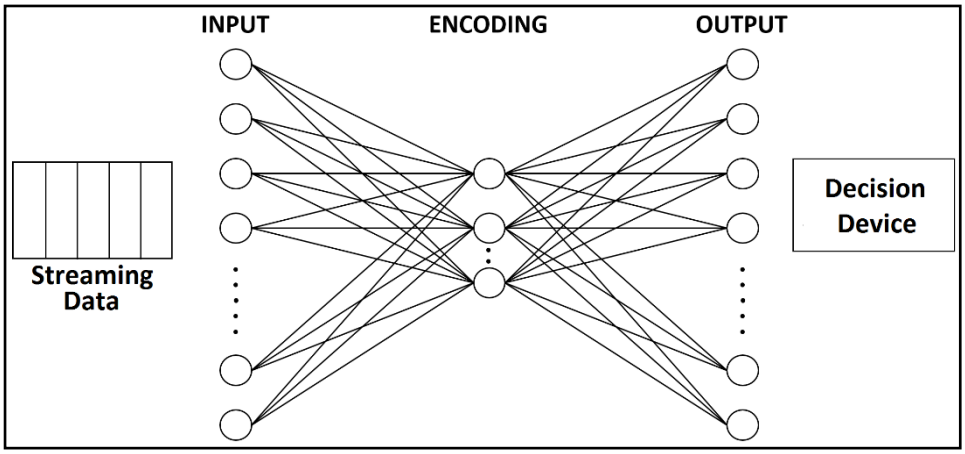


Figure 3.2: *Auto-Encoder - Basic structure*

In this section we take a short overview of auto-encoders. Auto-encoders are simple neural networks with one input, one hidden and one output layer. Hidden layer is also called as encoding layer. Input and Output layers have got same number of neurons while hidden layer has lesser number of neurons. We try to reconstruct input at the output layer. Now coding layer, as it has got lesser neurons, acts as a bottleneck. So it becomes important for coding layer to learn the concept instead of memorizing the input. Now if we train network with input from only target class, it will be able to reconstruct objects from target class much better compared to outlier objects. So error in reconstruction can act as a measure for membership of an object to target class. Decision device is tasked with calculation of this membership. This is basic concept behind auto-encoders. Accompanying figure explains it.

Chapter 4

Detailed Design and Architecture

This chapter will focus on details of architecture and design of system. It will start with high level overview of system and then it will take you through details of each and every component explaining all its aspects.

4.1 Basic design and Architecture

Now as we have developed a roadmap for application design and decided which technologies and methods to use, we are in situation to delve deeper into the design and architecture of application. So let us start by getting high level overview of application design. System architecture can be seen in following diagram:

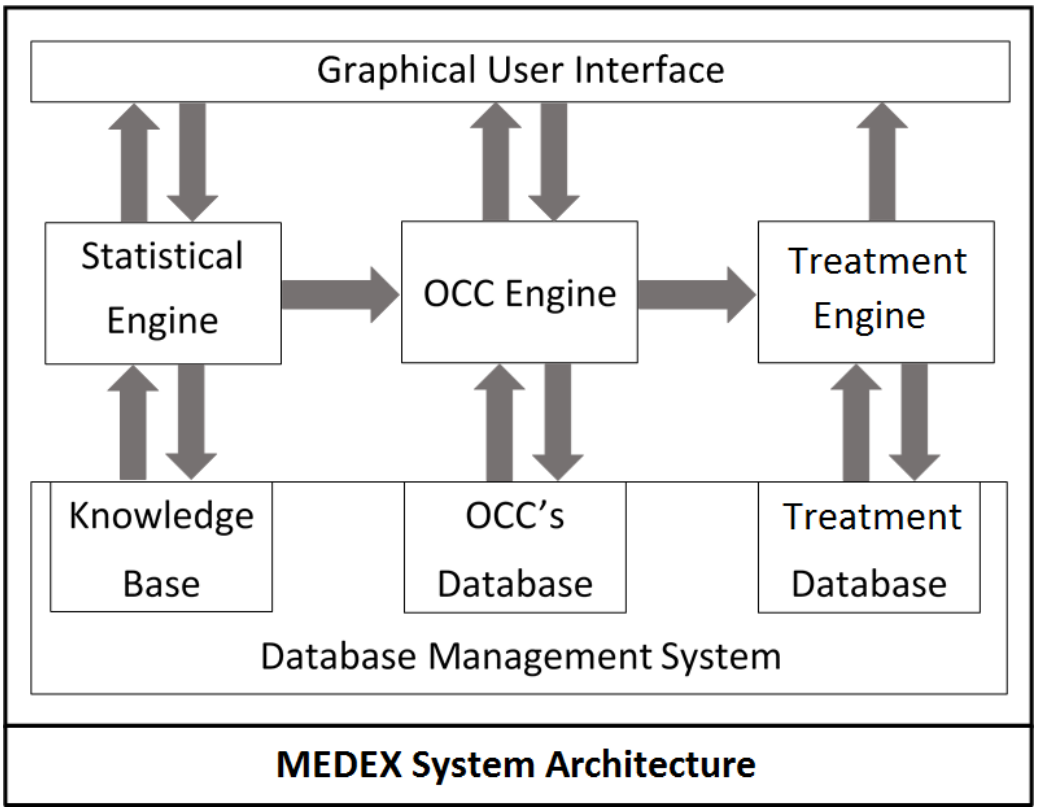


Figure 4.1: *System Architecture for MEDEX*

User will interact with system through GUI. He will provide system with general information as age and gender of patient and then will tell about symptoms faced by patient. Now, this input is processed by Statistical Engine (SE) with the help of Knowledge Base (KB) which is simply a database. KB stores past history about disease-symptom relationships. After processing the input Statistical Engine (SE) generates the list of most possible diagnoses say "Intermediate List" which it passes to OCC Engine (OE). OCC Engine finally runs OCC's corresponding only to diseases in the list obtained from SE (remember that we have separate OCC for each disease!) and produces final list of all possible diagnoses along with confidence factors. Now user selects the correct diagnosis from generated list. And this selection is used as a label for input data i.e. we tell system that this particular input corresponds to this disease. So this labelled input now acts as a new training set for OCC corresponding to that particular disease. It is also used to update entries in KB. This constitutes the learning phase of application. All data about OCC's is stored in

OCC database (OD). Now after user selects correct diagnosis, application provides latest generic treatment options available for that disease. This treatment is generated by Treatment Engine (TE) with the assistance from Treatment Database (TD). All three – KB, OD and TD are implemented with database management system (DBMS).

In current scenario, statistical engine looks like a useless stuff. After all final list of diagnoses is generated by OCC engine. So then why not use OCC engine alone? Why even get an intermediate list from statistical engine and then run OCC for diseases only in that list? Think about it. We have several hundred different diseases and hence same number of OCCs each corresponding to a separate disease. So we will end up trying out all the possibilities exhaustively. It becomes pointless to run all this hundreds of OCCs for every input as it is time consuming and resource intensive task. Also in most cases we will end up trying out completely illogical possibilities. E.g. we will try out OCC for heart attack even when patient says he has knee pain. This will waste time and is not at all suitable for real-time systems. Statistical Engine solves the very problem with great efficiency. This is how we are tackling the problem by perfectly blending statistical methods with machine learning and using best from both of them.

Now that we understood the overall architecture at high level, it is time to see how these components accomplish assigned tasks. We will look at detailed design and working of each component in next section.

4.2 Detailed design of each component

4.2.1. Input

Input consists of general information about patient like age and gender followed by symptoms faced by them along with their intensities. So each symptom can be seen as a four tuple as : < description of symptom, intensity of symptom, age of patient, gender of patient>. (< s, i, a, g>). And input is set of such four tuples.

4.2.2. Knowledge Base

Knowledge Base (KB) is simply a database tasked with assisting statistical engine. ER diagram can be seen on next page. It stores all data about disease-symptom relationship. It stores count of how many times each input tuple appeared, each disease appeared and how many times was particular disease diagnosed for particular input tuple. This data gives rise to statistical distribution for disease-symptoms relationship. This distribution is then used as a foundation for statistical engine.

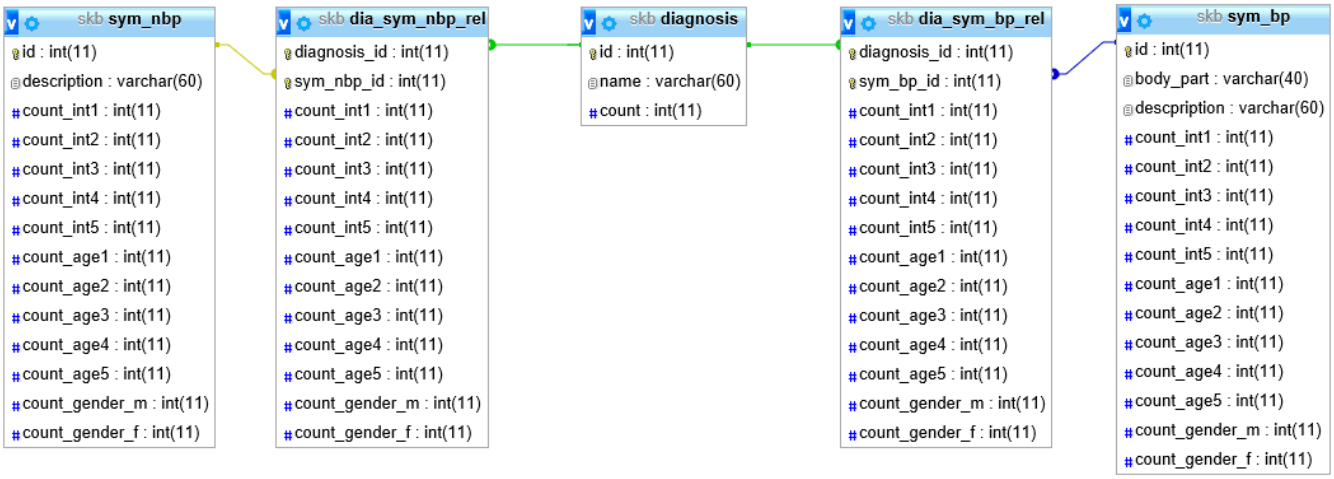


Figure 4.2: *ER diagram for Knowledge Base*

Let us take an example to make things clearer. Consider symptom (s) head ache. Also consider diagnosis (d) malaria. As you know head ache is one of the symptoms of malaria. But intensity of head ache can vary from case to case. Say intensity values span the range from 1 to 5. Now KB will contain count of how many times s and d appeared in past and how many times d was diagnosed for s with each intensity. So KB will look like –

Disease	Count
d	50

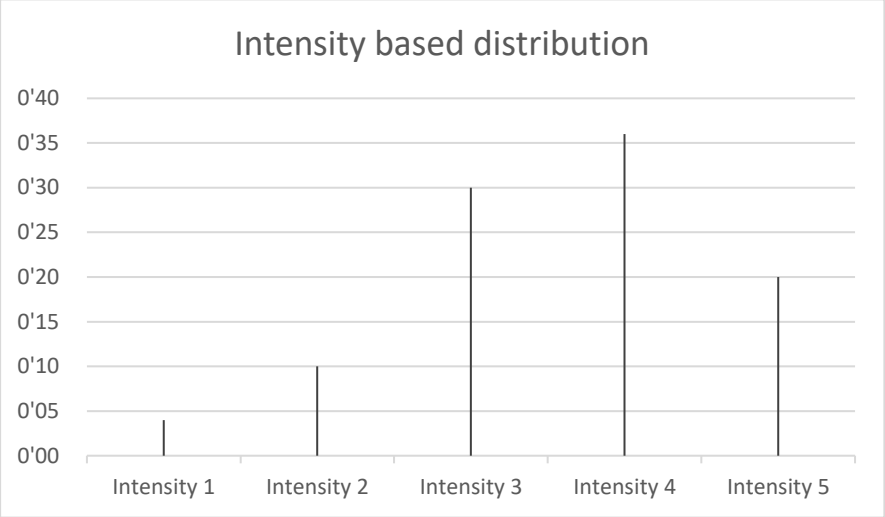
Symptom	Count
s	200

Disease-Symptom Relation	Count				
	Intensity 1	Intensity 2	Intensity 3	Intensity 4	Intensity 5
d-s	2	5	15	18	10

50

From KB, we can derive an intensity-based distribution for d-s relation. We can divide count of Intensity 1 i.e. 2 by total count of d i.e. 50. That gives us 0.04. We will call this intensity fraction for 1. Similarly, we get 0.10, 0.30, 0.36 and 0.20 as intensity fractions for 2, 3, 4 and 5 respectively and so we get the corresponding graph shown on next page.

Now, similar distribution can be obtained for age and gender. This distributions form basis for the working of Statistical Engine.



4.2.3. Statistical Engine

Main purpose of statistical engine is to act as a selector for OCC's i.e. provide list of most possible diagnoses to OCC engine. It is also tasked with generation of missing symptoms (refer step 3 part in Chapter 2 - System Overview for definition of missing symptoms) for each possible disease. Let us see how it accomplishes these tasks.

As stated above, KB basically builds a distribution for disease-symptom relationship. Consider input set containing single tuple (t) - <head ache, 3, 34, male> (description of symptom, intensity, age, gender). Head ache is one of the symptoms of Malaria (d). Now KB will have distribution corresponding to Head Ache (s) – Malaria (d) pair. It will also have count of how many times 'd' and 's' appeared in past. Now based on this distribution, we must decide how much 't' contributes to diagnosis of 'd'. For example, from above graph, we can notice that malaria is diagnosed maximum times with intensity value of 4 and least with intensity value 1. So if input intensity value in t is near 4, it should contribute more to diagnosis of malaria than if it is near 1. Same is the case with age and gender. In short we should assign weights to each input tuple-diagnosis pair that will tell how much input tuple contribute to that diagnosis. This is first step in Statistical Engine. Let us look at it in more details:

For weight calculations, we calculate three weights corresponding to intensity, age and gender distribution separately and then multiply them together to get total weight. We will look at calculation of weights corresponding to intensity as for age and gender, calculations are similar.

While calculating weight, we need to consider few things as follows.

1. Weights for value must be proportional to fraction in distribution at that value.
2. If distribution is equal, all values should get higher weights.
3. Closer is value to peaks in distribution, better should be its weight.

Let's try to calculate weight for input tuple and distribution graph given above. Input intensity is 3. Now consider intensity value 1 in distribution. We find following things at that point:

Fraction *= Intensity fraction at current point = 0.04*

Intensity gap *= modulus (Intensity at current point – Input Intensity) = modulus (1 – 3) = 2*

Fraction gap *= modulus (Fraction at current point – Fraction at input intensity)*

= modulus (0.04 – 0.30) = 0.26

Now we multiply all three quantities above to get a new quantity – total negative fraction at 1.

Total negative fraction at 1 *= (Fraction)*(Intensity gap)*(Fraction gap) = 0.04*2*0.26*

= 0.0208

Similarly, we can get Total negative fraction at all other intensities.

Total negative fraction at 2 *= 0.10*1*0.20 = 0.0200*

Total negative fraction at 3 *= 0.30*0*0 = 0.0000*

Total negative fraction at 4 *= 0.36*1*0.06 = 0.0216*

Total negative fraction at 5 *= 0.20*2*0.10 = 0.0400*

Now we add total negative fractions at all intensities to get net total negative fraction as 0.1024 and then we divide net total intensity fraction by 4 as it is maximum value it can take and then subtract it from 1 to get weight corresponding to intensity (wi).

wi = 1 – 0.25(Net total negative fraction) = 1 – 0.25*0.1024 = 0.9744.*

Similarly, we can calculate weights corresponding to age (wa) and gender (wg). Then we multiply them to get total weights (wt).

*wt = wi*wa*wg*

As said earlier, input is set of many symptom tuples. Now for each possible diagnosis, we get *wt*. So basically, for each diagnosis we have array of *wt* with each element corresponding to different input symptom. So now we are in position to calculate ***confidence factor*** for each possible diagnosis. Before we proceed let us define history factor for disease-symptom relationship. We define it as ratio of number of times disease d was diagnosed to number of times symptom s was seen. In current scenario it is 50/200 = 0.25.

Confidence factor has two components – ***positive CF*** and ***negative CF***. For ***positive CF***, we multiply history factor with total weight for each symptom and then sum it up for all input symptoms tuples and divide it with total number of symptoms.

$$\textbf{positive CF} = \frac{\sum_{\textbf{All Symptoms}} (\boldsymbol{Weight}) * (\boldsymbol{History\ factor})}{\boldsymbol{number\ of\ symptoms}}$$

In next step, we generate list of missing symptoms by looking at KB. If some symptom appears lot of times in diagnosis of particular disease, it can be thought as an important symptom for that disease. If user haven't provided that as an input symptom, it becomes missing symptom for that disease. Now we repeat same procedure of calculating weights and history factor assuming that missing symptoms are present. Then we proceed the same way we did for calculating positive CF. But now as these symptoms are actually missing, that becomes ***negative CF***.

Now finally, we are ready to get ***confidence factor (CF)***.

$$\textbf{CF} = \frac{(\boldsymbol{number\ of\ symptoms}) * (\boldsymbol{positive\ CF}) - (\boldsymbol{number\ of\ missing\ symptoms}) * (\boldsymbol{negative\ CF})}{(\boldsymbol{number\ of\ symptoms}) + (\boldsymbol{number\ of\ missing\ symptoms})}$$

This way we get confidence factor for each possible diagnosis. With this confidence factor, we choose the most probable list of diagnosis as triggering list for OCC's.

4.2.4. OCC Engine

OCC Engine is next step in process of diagnosis. As we saw, we have separate OCC for each diagnosis and Statistical Engine tells us which OCC's to trigger. Also as analyzed earlier, auto-encoders was the best choice. But every existing model, assumes the fixed size input vector. That limitation obviously renders the model useless for us as we have dynamism as inherent property of diagnosis process. We can't bypass the problem of variable input simply by mapping the input variable to fixed number of variables as we will lose a lot of data in process and it will make the diagnosis very difficult. So we modified an existing back-propagation method so that it can handle the dynamic input vector. Let us look at details.

First, we associate each symptom to particular input and output units and use symptom's intensity values as input values. Now, we reconstruct the output and find the Euclidian Distance between input and output vectors. Then we process it to get the class membership and confidence factor for each test case. While modifying existing model for dynamic input handling, we must keep in mind the following points –

1. Model must accommodate new inputs without affecting stability of other trained inputs and network as otherwise system will swing between stable and unstable states continuously rendering it useless.
2. Inclusion of new input must not make the network forget previous training.

3. Training must not require storage of previous training sets as they will tend towards millions very quickly.

While developing model, one very useful principle is that absence of some symptom can be considered as presence with zero intensity. We will use this principle a lot in development of our model.

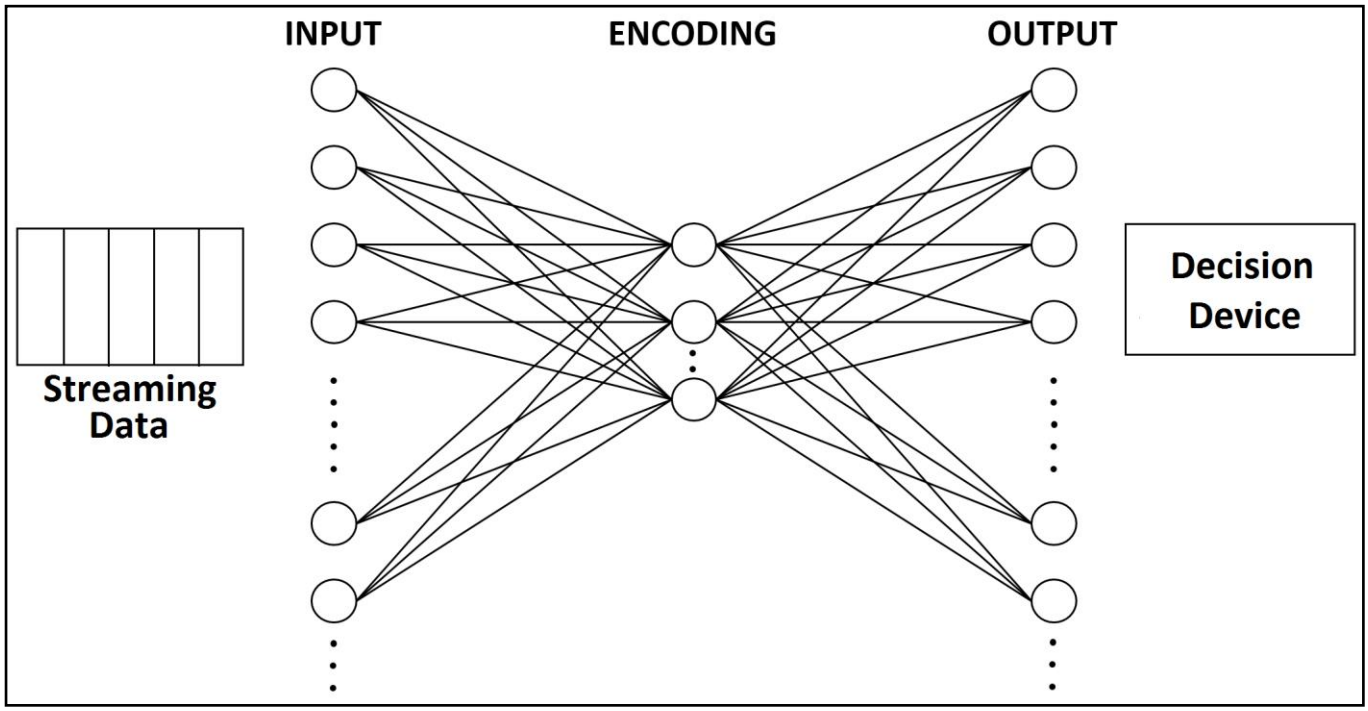


Figure 4.3: *Auto-Encoders with infinite inputs*

As number of input is variable and ever increasing, we assume infinitely many inputs (Refer the figure 4.3). We assume that there are infinite input nodes for which we don't know the association with symptoms right now. When we find some new symptom, we associate it to some empty input node. But once this association is done, it must be maintained in all the following test cases. Let's take an example to explain this. First consider completely naïve, untrained network. It has infinite input and output nodes but we don't know, any association i.e. we don't know which input and output correspond to which symptom neither we know to which they will correspond to, in future. Now suppose patient comes with head ache and fever as symptoms and doctor diagnoses influenza. Now, in OCC corresponding to influenza, we assign first input and output nodes to head ache and second to fever. But right now, we don't know what third input node corresponds to. It can in future be assigned to chills or to body ache or anything else. Now consider another patient diagnosed with influenza and having symptoms as fever and body ache. Now, we already have an association for fever and that will be maintained. But now we have found new symptom related to influenza which is body ache. So we assign it to next unassociated input and output nodes i.e. to third. This way we keep adding new inputs and training the network.

Even if we assume infinite inputs with unknown association, it is computationally perfectly feasible and we now explain how. As stated above, absence of symptoms can be treated as presence with zero intensity. So even if association of symptoms with inputs is unknown, there value is known to be zero and so there contribution to next i.e. encoding layer is also zero. In above example, we don't know association for fourth. It can be leg ache or back ache or anything else. But no matter what it may be, we know for sure that because patient is not having it, its intensity and hence the input value must be zero. That basically means that we don't need to bother about their calculations. This keeps the calculations limited even for infinite input size.

When new symptom is found, we initialize its encoder weights i.e. weights from input to hidden layer with one and decoder weights i.e. weights from hidden layer to output layer with zero. Then we train network with input. If some symptom is unavailable, like for second patient in above example head ache is unavailable, its value and hence contribution to next layer becomes zero. This prevents instability due to very small training of rarely or falsely appearing symptoms and noise taking care of stability swings.

As stated earlier, we need to make training online. But it is very difficult to make backpropagation algorithm online sequential. So after new training instead of replacing previous weight vector with new one, we move it into direction of new weight vector with step size inversely proportional to number of total trainings done till now. This helps in ingesting new training example without forgetting previous training completely. This way, we accomplish all the three aims mentioned above.

4.2.5. OCC Database

This simply is a database which assists OCC Engine in its functioning by storing all data about OCCs like – weights and metadata.

4.2.6. Treatment Engine

It is tasked with providing generalized treatment based on diagnosis selected by the user. This engine simply searches the treatment database which is static database for treatment corresponding to given disease. We don't use any learning or statistical algorithms here as prescribing medicines is lot more complex than identifying disease. We maintain a static manually updated and preloaded with fixed generic treatment (i.e. we prescribe drugs and not medicines) for each disease. Providing particular treatment by considering all complexities like drug allergies, medical history, economic factors etc. is left as a future scope.

4.2.7. Treatment Database

It is simplest component which is a database with single table which stores treatment plan for each disease. It is fixed and manually updated every time a new treatment is to be added or any treatment is to be updated.

Chapter 5

Implementation details

Chapter aims at exploring the technologies used for implementation of various component and routines used in application.

Now let's take a look at technology stack used to implement the discussed application.

5.1 Front End Development

As you can see, system is trained every time doctor uses it for diagnosis. So, more is system used, more trained it is and better it gets. So we developed it on web platform as a browser based application so that it can be used by many users at same time and still they will share the Knowledge Base. This means, Knowledge Base enriched by knowledge from expert doctors will be accessible to newbie practitioners. This will help a lot in sharing of expertise. Developing browser-based application also helps to make it cross-platform as well as responsive to various screen sizes.

We used HTML5 with CSS3 for front end development. For CSS, we used W3 framework. We also used JavaScript (jQuery version 1.11.3) for interactive design.

5.2 Back End Development

We have implemented all the previously discussed algorithms and subroutines using core PHP (version 7.0.6) as we are developing it as a web-based application. For running backend codes, we used XAMPP server (version 7.0.6) on localhost. Developed application works with two MySQL databases (version 10.1.13), corresponding to statistical engine and OCC engine respectively.

So finally this is summary of technology stack –

1. HTML5, CSS3, jQuery (1.11.3)
2. Core PHP (7.0.6)
3. MySQL (10.1.13)
4. XAMPP server (7.0.6)

Chapter 6

Experimentations, Results and Conclusions

This chapter gives details about data collection, data mapping and experimentations. It finally provides the results of experimentation and concludes with remarks.

The problem of automating medical diagnosis has not been touched upon till now. So there are no readymade training and testing datasets available for this problem neither are there any existing software to compare with. All existing software use static databases which are preloaded by some expert. These software are not capable of handling dynamism on their own and requires human expertize for that. This drawback is tackled with in our application.

6.1 Data set preparations

Due to unavailability of training and testing data, we had to collect our own data by sitting in cabin of medical practitioner and noting down symptoms and diagnosis for each patient. This real time data collection was a major challenge. We decided to collect data for two most general and frequent diagnoses – influenza and common cold. We could get only 46 cases for influenza and 59 cases for common cold. This data was not at all sufficient for training and testing. So we studied the trends in the data. We analyzed frequency of different symptoms. From that we could understand the significance of various symptoms in diagnosis of that particular disease. Then we mapped them to 920 and 1180 test cases by randomly mapping symptoms 20 times but still maintaining the original ratios and trends. We used these cases for training and the data collected in real time was used as testing cases.

6.2 Concept of accuracy

Also as stated earlier in overview section, instead of providing single firm diagnosis, our application floats out list of most probable diagnoses along with confidence factor for them and then doctor choses the correct one from that list. This requires different measure for accuracy. We will tell in how many test cases correct diagnosis was ranked first in the list, second in the list and above second in the list. More is the percentage in first ranking, better is the accuracy. But second rank is also quite acceptable.

6.3 Results

As stated above, we provide results with three labels in following tables:

Results for Influenza Testing (Total cases : 46)		
Rank of Influenza in list	**Number of test cases**	**Percentage of test cases**
First	27	59%
Second	14	30%
Third onwards/ absent	5	11%

Results for Common Cold Testing (Total cases : 59)		
Rank of Common Cold in list	**Number of test cases**	**Percentage of test cases**
First	34	58%
Second	16	27%
Third onwards / absent	9	15%

Notice that for both the diagnoses, correct disease was ranked first for about 60% of the times. It was ranked second in about 30% of the cases. In remaining 10-15% cases, it was either missing or ranked third onwards. So we can conclude that overall accuracy of system is very good considering the complex and dynamic nature of field of diagnosis which lacks any kind of mathematical modelling. Moreover this accuracy makes system suitable for use as on average accuracy of even human intelligence is only about 70% in diagnosis process. With this we conclude the chapter.

Chapter 7

Benefits and Scope for future work

This chapter throws light on benefits of developed application. It also discusses what can be done to improve current application and make it ready for deployment.

Now that we took you through what we have done, it's time to highlight what good the developed application will do. This application will prove to be very useful for medical practitioners. Here are its benefits:

1. It provides them with list of all possible diagnoses and hence eliminates possibility of missing some diagnosis due to ignorance or just because they don't have perfect memory and can't remember everything.
2. After selecting correct diagnosis application also provides a high level generic treatment. That can help them in refreshing their memory before prescribing drugs to the patient. It also helps them in getting acquainted with new treatment options very easily. This can save them a lot of time and work.
3. The application will also help in sharing expertize between doctors. It will get trained by many doctors as it is built on distributed platform and now a newbie practitioner will be using same knowledge repository developed by thousands of expert doctor.

Moreover application will also help in reducing malpractices seen commonly in field of medical diagnosis and expand its reach to under-privileged sections in community.

Scope of automation in many fields is restricted due to inability of learning algorithms to handle dynamism which is inherent to that field. Field of medical diagnosis is no exception. While developing this application we devised new methods for effectively tackling problem of dynamic size of input and output vectors with the use of statistical algorithm and modifying the existing model of auto-encoders. **This project will act as a small step towards adopting learning algorithms in problems with inherent dynamism which is much required in current scenario.**

Though we have tried to take care of as many things as possible in this project, it will always remain open for further development. We can introduce Natural Language Processing (NLP) module as very first step in diagnosis to understand symptoms in much better and more humanly way. That is a huge scope for future work. Similarly, we can introduce deep learning network after extracting features with the help of auto-encoders to integrate age, gender and other factors more naturally than they are now. User accounting system can also be integrated to keep track of past medical history of user and use it in subsequent diagnosis and treatment. At last, treatment part of application needs tremendous development if we want to provide particular treatment instead of generic one based on diagnosis, medical history of patient, drug allergies and many other factor. This is lot of scope for future work.

References

1. Tax, D.M.J., 2001. One-class classification: concept-learning in the absence of counter-examples [Ph. D. thesis]. *Delft University of Technology, Stevinweg, The Netherlands*.

2. Gautam C., Tiwari A., and Leng Q, "On The Construction of Extreme Learning Machine for Online and Offline One Class Classification - An Expanded Toolbox, *Neurocomputing (ELSEVIER) (Accepted)*

3. Manevitz, L. and Yousef, M., 2007. One-class document classification via neural networks. *Neurocomputing*, *70*(7), pp.1466-1481.

4. Irigoien, I., Sierra, B. and Arenas, C., 2014. Towards Application of One-Class Classification Methods to Medical Data. *The Scientific World Journal*, *2014*.

5. Leng, Q., Qi, H., Miao, J., Zhu, W. and Su, G., 2015. One-class classification with extreme learning machine. *Mathematical Problems in Engineering*, *2015*.

6. Mazhelis, O., 2007. One-class classifiers: a review and analysis of suitability in the context of mobile-masquerader detection. *Arima Journal*, *6*, pp.29-48.

7. Khan, S.S. and Madden, M.G., 2009, August. A survey of recent trends in one class classification. In *Irish conference on Artificial Intelligence and Cognitive Science* (pp. 188-197). Springer Berlin Heidelberg.

8. Khan, S.S. and Madden, M.G., 2014. One-class classification: taxonomy of study and review of techniques. *The Knowledge Engineering Review*, *29*(03), pp.345-374.

9. Schölkopf, B., Williamson, R.C., Smola, A.J., Shawe-Taylor, J. and Platt, J.C., 1999, December. Support Vector Method for Novelty Detection. In *NIPS* (Vol. 12, pp. 582-588).

10. Japkowicz, N., Myers, C. and Gluck, M., 1995, August. A novelty detection approach to classification. In *IJCAI* (Vol. 1, pp. 518-523).

11. THE WASHINGTON MANUAL OF MEDICAL THERAPEUTICS (34th Edition) Department of Medicine, Washington University, School of Medicine, St. Louis, Missouri

End of the Thesis