12-25-2018

Predicting Post-Procedural Complications Using Neural Networks on MIMIC-III Data

Namratha Mohan
Louisiana State University and Agricultural and Mechanical College

PREDICTING POST-PROCEDURAL COMPLICATIONS USING NEURAL NETWORKS ON MIMIC-III DATA

A Thesis

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science

in

Department of Computer Science

by
Namratha Mohan
B.E., Visvesvaraya Technological University, 2011
May 2019

Acknowledgments

First and foremost, I would like to express sincere gratitude to my advisor Dr. Thanos Gentimis for being the best advisor I could have asked for. His invaluable insight, guidance and support throughout the research journey has helped me become a better student.

I would like to thank Dr. Konstantin Busch for his help, advice and support throughout my work and for agreeing to serve as my MS thesis committee chair. Special thanks to Dr. Bijaya B. Karki for agreeing to serve on my thesis committee and contributing his time.

Most importantly, I would like to thank my parents Mohan Iyer and Sukanya Mohan, and my friend Pavan Chandrashekar for letting me follow my dreams and for their love and encouragement which has been a vital part of this research.

Table of Contents

List of Tables

List of Figures

Abstract

The primary focus of this paper is the creation of a Machine Learning based algorithm for the analysis of large health based data sets. Our input was extracted from MIMIC-III, a large Health Record database of more than 40,000 patients. The main question was to predict if a patient will have complications during certain specified procedures performed in the hospital. These events are denoted by the icd9_code 996 in the individuals' health record. The output of our predictive model is a binary variable which outputs the value 1 if the patient is diagnosed with the specific complication or 0 if the patient is not. Our prediction algorithm is based on a Neural Network architecture, with a 90%-10% training-testing ratio. Our preliminary analysis yielded a prediction accuracy above 80%, outperforming various multi-linear models [2]. A comparative analysis of various optimizers as well as time based performance measures is also included.

Chapter 1.
Introduction

The use of neural networks to predict complications when certain specified procedures are performed during the patient's hospital stay, has not been attempted in the past as per the literature review. Still, various computer assisted diagnostic (CAD) tools created lately attempt to give similar answers, and some of them are based on ideas from Machine Learning and specifically Deep Learning. It is imperative thus, to explore the question both from a computer science perspective but also from a health informatics one.

Predicting post-procedural complications would be beneficial for all stakeholders including hospital clinicians, doctors, administrators and of course the patients and their families. For clinicians, these models can help in categorizing patients, allowing them to take precautions and better manage those patients that are deemed more likely to have complications. Administrators, would also find this an invaluable tool to justify the extra expenses needed to manage these patients in a post-procedure hospital stay and help with hospital resource management. The main benefit obviously, is the fore-knowledge that the patient would receive, allowing them to be more mindful. Avoiding complications that can be avoided, would shorten their stay at the hospital and help the recovery process.

The general icd9 code 996 defines complications particular to certain specified procedures. Multiple categories fall under this code such as complications due to cardiac device, vascular device, nervous system device, genitourinary device, complications of specified prosthetic device, infection and inflammatory reaction due to internal prosthetic device, complications due to internal orthopedic device and transplated organs and others. Each of these categories subdivides further to different types of complications associated with it [5].

Neural networks are used primarily for classifying information, predicting outputs and clustering. Specifically, many applications of them exist in pattern recognition [27], image processing, forecasting, classification [12] and others. Their use in health care ranges from

clinical diagnosis, image analysis and interpretation, to signal analysis, including drug development and more [6].

In our work, we are primarily using MIMIC-III [28], which is a medical database containing de-identified health related data of over forty thousand patients admitted in the critical care units of the Beth Israel Deaconess Medical Center between 2001 and 2012. [1]

After the appropriate pre-processing phase, four different neural networks were created using the keras package each with a different optimizer and averages over multiple runs were computed. Although we did not monitor the memory allocation, we compared the optimizers both on accuracy and time and discussed the results.

This thesis is organized as follows: In Chapter 1 we introduce the problem and discuss it's relevant parts. In Chapter 2, we analyze relevant literature and motivate the thesis. In Chapter 3 we describe the experimental setup including the system hardware, programming language, database, and the specific machine learning tools used. In Chapter 4, we talk about the implementation, covering the pre-processing and machine learning model in detail. Following that, we discuss the experimental results and the prediction accuracy in Chapter 5. Finally, we present out conclusions and future work in Chapter 6.

[1] https://mimic.physionet.org

Chapter 2.
Literature Review

2.1. Health Informatics

Health Informatics is the collection of practices, algorithms and procedures for acquiring, maintaining, retrieving and analyzing healthcare information. It makes use of information technology to organize and analyze health records of patients, to improve various healthcare outcomes with primary focus being that of patient care [9].

An Electronic Health Record (EHR) contains the digital version of a patient's health records. This includes information related to the patient's demographic details, progress notes, medications, vital signs, immunizations, past medical history, laboratory data and so on. Since their establishment, EHRs have bridged the gap between patients and clinicians allowing transparency and transferability . Various vendors who provide the patients with such EHRs are Epic Systems Corporation, Allscripts, NextGen Healthcare, GE Healthcare, Cerner Corporation, CureMD, practoce fusion, eClinicalWorks and so on.

New methodologies promoting the use of EHRs and cross platform interoperability have been on the forefront of Health Informatics the past few years. This push has been supported by the need for "meaningful use" of the Medicare and Medicate mandates established during the latest health reform act. De-identified Electronic Health records for research purposes have thus become abundant and more and more researchers from various fields have been interested in analyzing and interpreting them.

2.2. Neural Networks General

Warren McCulloch, a neurophysiologist and a mathematician Walter Pitts created the first neural network model in 1943 [16]. Their model initially tried to explain how neurons in the brain work, and it was purely based on mathematics and logical algorithms. Later on, in 1949, D.O Hebb introduced the concept of "Hebbian learning" which was a learning hypothesis based on the neural plasticity and is now explaining the intuitions behind the

architecture of NN's. In 1958, Frank Rosenblatt created the Perceptron, which is the key element in basic NN designs and was the first ever model that could do pattern recognition. Unfortunately, his model couldn't be tested due to lack of information.

The first "real" neural network model with many layers was first tested and published by Alexey and Lapa in 1965. Paul Werbos introduced the Backpropogration concept which gave a solution to the XOR problem and made neural networks more efficient. More recently, between the years 2009 and 2012, the deep feedforward neural networks and recurrent neural networks that were developed by Schmidhuber won eight competitions related to pattern recognition and machine learning [18]. The field of Machine learning, which NN's are the primary star, is now grown to include experts from various fields and one can argue that it is on the forefront of Data Analytics this past decade.

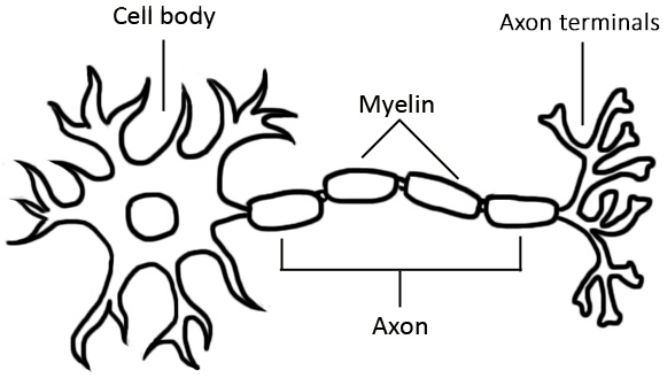


Figure 2.1. Working of a Neuron

In many cases, rigid algorithms can do well what humans can do but even then, human brains are seemigly always a step ahead. An Artificial Neural Network mimics the functionality of the structure of human brain in solving machine learning problems and acts more like a human brain instead of a series of elaborate "if then" statements [31]. As shown is Figure 2.1 [30], our brain comprises neurons or nerve cells which are used in transmitting and processing information received from our senses. Multiple neurons are

arranged together to form a network passing electrical impulses from one neuron to the other.

The dendrites receive these electrical impulses from the synapses of the previous neuron which is further carried to the soma (nucleus of the nerve cell). The electrical impulse is processed in the soma and the output is then transferred to the axon. These axons are the longer branches which transfer the electrical impulse from the soma to the synapse. These synapses will later transfer the impulse to the dendrites of the second neuron [32]. One is amazed when he/she realizes that the exact same concept is used in the working of neural networks as we will discuss further below.

We note here that the adoption path of neural networks is unique when compared to other technologies since they neither failed immediately nor became hits overnight but instead their popularity rose gradually and remained the same ever since then.

2.3. Predicting Post Procedural Complications

As mentioned earlier, to our knowledge there is no neural network based model that uses MIMIC-III Data to predict post-procedural complications during a patient's hospital stay. Some scientific papers attempt to predict post-procedural complications but are limited to a particular health condition and/or use classical statistical tools.

On example of special focus is the paper "Prediction of Sepsis in the Intensive Care Unit using MIMIC-III Data" by Thomas Desautels et al., [20]. The main purpose of this article is to test a machine learning based system called *Insight*, in terms of predicting the onset of sepsis based on the new Sepsis-3 definitions using a minimal set of electronic health record variables from the MIMIC-III database. The performance of Insight is compared with the already existing scoring systems: quick sequential organ failure assessment (qSOFA), modified early warning score (MEWS), systemic inflammatory response syndrome (SIRS), simplified acute physiology score (SAPS) II, and sequential organ failure assessment (SOFA) to find whether or not the patients will become septic at a fixed period of time. Additionally, the effects of data sparsity is investigated on the Insight performance.

Another example is the paper "Predicting Hospital Length of Stay using Neural Networks on MIMIC-III Data", in which the authors attempted to predict the total length of stay of a patient in hospital using various patient and admission data provided by the MIMIC-III database [24]. Three different models were run on the data: neural network, random forest, linear model. The neural network provided the most accurate results.

In their paper "A prognostic computer model to individually predict post-procedural complications in interventional cardiology", Budde et al., derived computer algorithms from artificial intelligence to predict the individual risk of post-procedural complications associated with coronary intervention and to explain the structural relationship between risk factors and post-procedural complications [14].

Chapter 3.
Experimental Setup

In this chapter we will give an overview of all the software and hardware used in this process, including the specific libraries and packages. We will also present an overview of the mathematical underpinnings behind the ideas of Neural Networks and give a brief description of the MIMIC-III database. Our goal was to create a "self-contained" document as much as possible.

3.1. System Hardware

The experimentation was carried out in a workstation that was configured as a server located at the LSU's Computer Science Department. All the required documents, databases and necessary computations were created and stored on that machine.

This server is an exceptionally powerful machine consisting of a Dual Intel Xeon Processor E5-2697 v2 (Twelve Core HT, 2.7GHz Turbo, 30 MB) and 64 GB of memory.

3.2. Python

Early in our research we identified that the framework of the analysis was more compatible with Python [2], which is an easy to learn object-oriented programming language. Python makes use of an elegant syntax, making it easier to read and understand scripts written in it. During our analysis we made use of multiple Python Libraries such as NumPy, Pandas, Keras and more. In the following subsections we describe those libraries in detail.

"Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications... " Guido van Rossum, the creator of the language explained in an interview about the language and its general philosophy.[1]

Python is the descendant of the ABC language which was conceived by Guido van Rossum at Centrum Wiskunde and Informatica (CWI) in the Netherlands in the late

[1]The Making of Python

1980s and is capable of exception handling and interfacing with the Amoeba operating system [35]. Its first implementation began in late 1989 and it is generally recognized that the Python syntax helps the programmers code in fewer steps when compared with other general purpose programming languages. The Python programming language is now widely used in bigger organizations due to the small learning curve and the plurality of features [19]. The vibrant community of python is also a big plus, since most problems and simple tasks can be addressed through the various python fora which are updated constantly.[2]

In the rest of the section we will discuss the various packages we used when implementing our codes.

3.2.1. NumPy

NumPy [3] is one of the most extensively used packages in Python with emphasis on scientific computing. It provides several features such as: computationally efficient and powerful multidimensional array object manipulation, linear algebra tools, Fourier transforms, random number generation algorithms and more. The most important object of NumPy is the homogeneous multidimensional array whose elements are most commonly numbers, indexed by positive integers.

3.2.2. Pandas

Pandas [4] is probably the most essential package for basic data analysis in Python. It provides high performing data structures and data analysis tools, with intuitive codes and extensive examples. Pandas allowed us to treat our dataset as a relational database table within Python. It is more effective to process and analyze ordered and unordered time series data with Pandas, than any other library and many statistical analyses can be easily performed with it. It provides two important data structures: series and dataframes; Series are responsible for handling single dimensional data whereas dataframes handle two and

[2]https://stackoverflow.com

above dimensional data (tables, tensors etc). Finally, It is built with many connections to NumPy making the integration with scientific computation seamless.

3.2.3. Keras

Keras [15] is an easy to use machine learning library built on Tenserflow [10] and Theano [11]. It provides many functional high level tools like Simple Neural Networks, Deep Neural Networks, Convolutional Neural Networks as well as various API's for developing and evaluating deep learning models. Keras is one of the most user friendly and "pythonic", semi-automated machine learning libraries out there. The main unit in Keras is the module which can be manipulated, extended and combined with other structures to match the specifics of the problem.

3.3. Stochastic Gradient Descent

The training of the neural networks utilized the idea of the "Stochastic Gradient Decent" (SGD) iterative method [17]. SGD calculates the error for each training example within the given dataset based on a set of parameters (weights) given to the inputs. Then the parameters are updated for each training example sequentially by a prediscribed choice vector. The updates of the parameters are then connected to the rate of improvement [21] and the best ones are chosen. The process repeats until a predetermined maximum (or minimum) is obtain on the error function, or the algorithm reaches a certain number of steps. Stochastic gradient decent is one of the most well known algorithms for training a variety of models in machine learning including logistic regression, support vector machines, simple linear regression, neural networks and others.

There are many implementations of SGD each with its own strengths and weaknesses. In our experiments, we made use of four optimizers: Adam, AdaGrad, RMSProp and AdaMax which we explore more below.

1) **RMSprop Optimizer** - Root Mean Square Propagation(RMSProp) [23] is an optimization algorithm which does well on non-stationary problems(noisy data) [26]. This

optimizer was proposed by Geoff Hinton.

2) **AdaGrad Optimizer** - AdaGrad stands for Adaptive Gradient. John Duchi, Elad Hazan and Yoram Singer first proposed this optimizer in their paper in 2011 [22]. This optimization algorithm is used to improve the performance on problems with sparse gradients.

3) **Adam Optimizer** - Adam [29] is an optimization algorithm to train deep learning models. Diederik Kingma and Jimmy Ba first came up with this optimizer in their 2015 paper. This optimizer combines the best results of RMSProp and AdaGrad to handle sparse gradients and noisy data [13].

4) **AdaMax Optimizer** - AdaMax [29] is a variant of Adam optimizer based on the infinity norm. Diederik Kingma and Jimmy Ba were the one's who came up with this optimizer. The infinite order norm makes the algorithm more stable which is best suited for sparsely updated parameters.

3.4. PostgreSQL Database System

PostgreSQL [1] is an open source object-relational database system. It is compliant with ANSI SQL Standards, which makes it easy to interact with most other database systems. In conjunction with SQL, Postgres also provides additional capabilities to store and query even the most complicated data workloads. Features like reliability, data integrity, extensibility (PostgreSQL provides the option to define our own data types) and most importantly the contributions from an open source community have made PostgreSQL one of the most powerful database systems available. It is supported by almost all the operating systems and is ACID compliant. [3]

PostgreSQL was first derived from the POSTGRES package written at the University of California at Berkeley and its first implementation is due to Professor Michael Stone-

[3]ACID stands for Atomicity (Each transaction is guaranteed either to succeed at once completely or to fail completely), Consistent(Data integrity is maintained), Isolation (Multiple transactions can occur concurrently without any data integrity issues) and Durability (once the transaction is completed, it will remain committed even in case of system failure).

braker (1986). Postgres has gone through several major releases since then. In 1996, a SQL language interpreter was added to Postgres by Andrew Yu and Jolly Chen. From then, Postgres was renamed to PostgreSQL to reflect the relationship between the Postgres and the newly implemented SQL interpreter. Postgres has been used in many applications ranging from data analytics, to various automated monitoring packages, including an asteroid tracking database, medical information databases, and several geographic information systems [1].

3.4.1. PgAdmin

PgAdmin [33] is an open source tool which is readily available in the PostgreSQL package. It is designed as a powerful graphical user interface for visualizing the database and making it easy to perform simple queries and data management, even for people with minimal coding skills. A complete write up with instructions detailing the process of creating a database through PgAdmin is included in the following section below.

3.5. MIMIC-III Database

In our work, we are making use of the MIMIC-III (Medical Information Mart for Intensive Care III) Database for predicting complications to certain specified procedures upon which the neural network is being trained. The MIMIC-III [28] dataset is a medical database containing de-identified health related data of over forty thousand patients admitted in the critical care units of the Beth Israel Deaconess Medical Center between 2001 and 2012 and is freely available for research purposes.

The MIMIC-III dataset contains the following information, structured in more than 20 tables:

1) Patients: Unique characteristics about each patient such as demographic details and more.
2) Admissions: Patient's specific hospital admission data.
3) Diagnoses: List of diagnoses performed on each patient and its corresponding details

such as icd9 codes(International Classification of Diseases Version 9) and more.

4) Procedures: List of procedures performed on each patient and its corresponding details such as icd9 code(International Classification of Diseases Version 9) and its descriptions and more.
5) Callout: Information about patient's Intensive Care Unit (ICU) summary.

As mentioned earlier, MIMIC is an openly available data set which is developed and maintained by the Massachusetts Institute of Technology(MIT) Lab. MIMIC-III database is updated periodically as more and more data becomes available. The updates to the database are made in batch and are given a new version number. MIMIC provides an in-house data querying tool for running SQL queries on the MIMIC-III database. The main purpose of this querying tool is to provide the researchers who are new to MIMIC with a light exploration of the data helping them better understand the structure.

3.6. Neural Networks

Artificial Neural Networks (ANN) or just Neural Networks (NN) are objectively the main tool in machine learning appropriate for handling large data sets. Neural Networks are a combination of "neurons" and "synapses" consisting of three main components: An input layer, a number of hidden layers and an output layer. These three parts create what is called an n-layer Neural Network. Each layer is connected with a set of weights and a bias value to the next one. Also, in each hidden layer a choice of activation function must be defined, but if that is fixed in the beginning of the analysis, only the weights and bias values will affect the output, thus training a Neural Network is a process of fine tuning the weights and bias values to get a better accuracy through a complicated Stochastic Gradient Descent method.

Every iteration in training the neural network contains two main steps: Backpropagation and Feedforward. Feedforward is the process of calculating the predicted output and Backpropagation is the process of updating weights and biases after a specified number of iterations [34].

Figure 3.1 below shows the architecture of a single layer perceptron. This is the simplest type of an artificial neural network, akin to the biological neuron presented in the literature review.

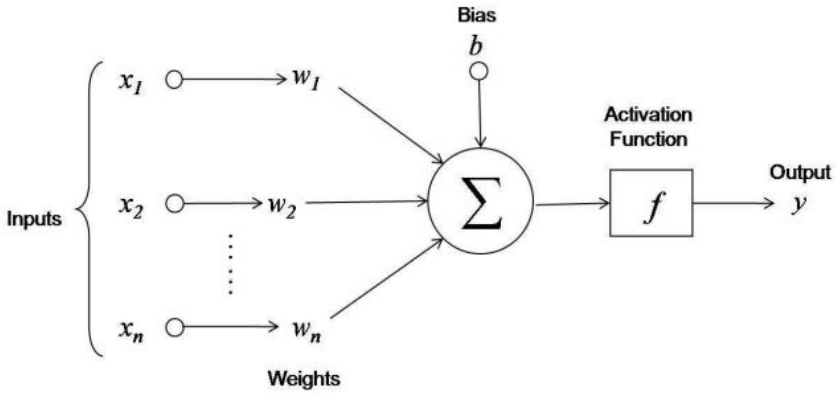


Figure 3.1. Single layer perceptron

Below are the major components of a perceptron:

1) Inputs: All the features available in the training dataset become the input for a perceptron. Also, an extra value called a bias value is fed as one of the inputs. In the above figure, inputs are represented by [x1, x2,.. xn] and the bias value is represented by "b".
2) Weights: The value of weights are initiated randomly (most of the times zero for all) and these values are updated accordingly by reviewing the training error. In the above figure, weights are represented by [w1, w2,.. w3].
3) Weighted sum: This is the summation of all the values obtained after multiplying each weight with its associated input value and adding the bias at the end.
4) Activation function: These functions convert an input signal of a node to an output signal. Some of the commonly used activation functions are tanh, sigmoid, relu, softsign, softplus, selu, softmax, elu, exponential and linear [7]. The flexibility of these activation functions is one of the reasons neural networks perform better than traditional multi-linear models.
5) Output: The weighted sum is passed into the activation function and becomes the input

value of the next layer.

As a first step, the weight vector is initialized. All the features available in the training dataset are fed as input to the perceptron. These input features are then multiplied with the corresponding weights and the values are summed up including the bias value. The new computed value is fed to the activation function in order to get the predicted output. If the predicted value doesn't match with the actual value, the error is calculated and the weights are updated in order to reduce the error for the next iteration. This process is repeated until the error is reduced to a prescribed level, or if a certain number of steps is achieved.

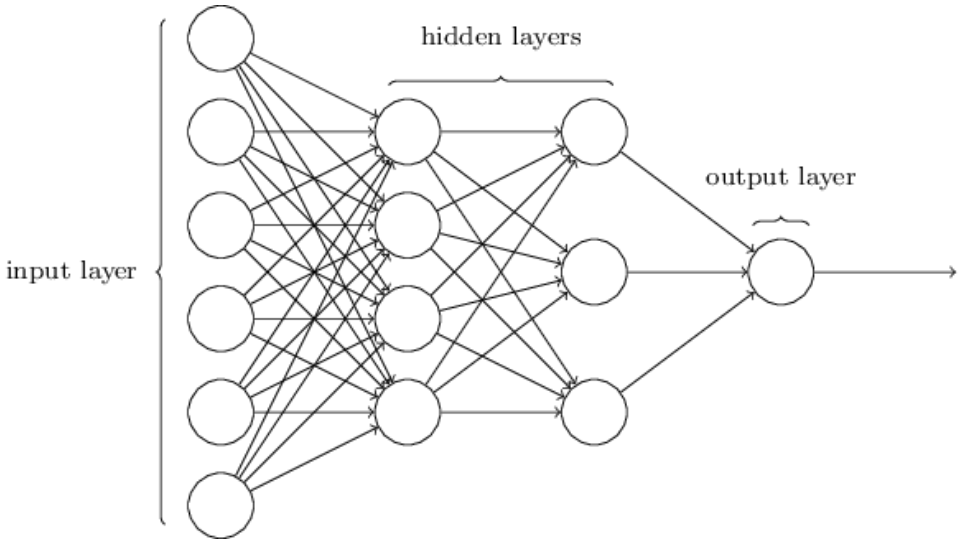


Figure 3.2. n-layer neural network

Multiple single layer perceptrons are stacked up into several layers to form a n-layer neural network [25] as shown in Figure 3.2.

Chapter 4.
Implementation

As we mentioned earlier, our main goal was to predict if a patient will suffer from complications after certain specified procedures are performed during their hospital stay. We also wanted to explore the performance of the various implementations of Neural Networks optimizers available to us with respect to predictability and resource allocation. In this section we will describe all the procedures, review our codes and explain the structure of each part of the algorithm.

4.1. Preprocessing

First we will describe the steps to set up a PostgreSQL-based clone of the MIMIC-III database on a local machine, as well as the queries we performed to clean up our dataset and make it suitable for the Neural Network implementation. As usual, real world data are often incomplete, resulting in missing values, noisy data, inconsistencies and so on. For us, data pre-processing is the collection of techniques that involves transformation of such raw data into an understandable format.

The process splits into the following steps:

4.1.1. Steps to create a database in PostgreSQL

The first thing one has to do is create a Database in PostgreSQL as follows:

1) Once the PostgreSQL software is installed on the local machine, we get a folder containing the following: Application Stack Builder, PgAdmin 4, Reload Configuration and SQL Shell (psql).
2) Open the PgAdmin 4 tool and expand the Servers followed by the PostgreSQL on the Left (It prompts to input a password and the default password is postgres).
3) Right click on the Login/Group Roles – Create – Login/Group. This is done in order to create a new username. It is always better to create a new username and password instead of using the default "postgres" username.

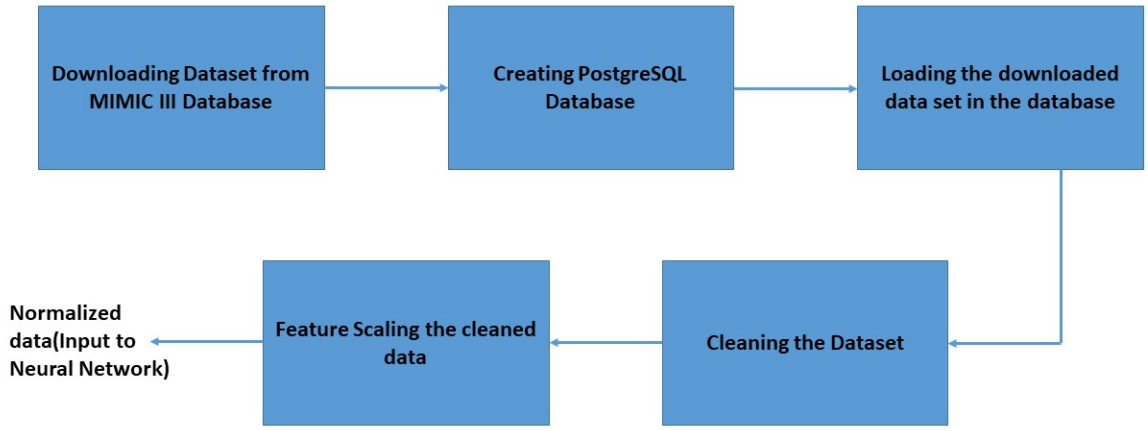


Figure 4.1. Describing the Preprocessing flow

4) Fill in the Preferred Name, move to the Definition tab and provide the desired password. Next, switch to the Privileges tab, change all the privilege settings to "Yes" and click on the Save button.
5) To create a new database, right click on the Databases and select – Create – Database.
6) Provide the Database Name, change the owner to the newly created username from the dropdown and click on Save.
7) Finally to create tables using PostgreSQL queries, right click on the Database name and click on the "Query Tool" option. The Query Tool will open on the right and through this the queries can be executed.

4.1.2. Steps to create our clean Dataset

After setting up the database, we describe the generation of our dataset by gathering input attributes from a combination of tables in the MIMIC-III Database listed below:

1) Admissions: hadm_id, subject_id, insurance, religion, martial_status, ethnicity, admittime, dischtime

2) CPTEvents: costcenter

3) ICUStays: first_careunit, last_careunit, los

4) Services: prev_service, curr_service

5) Patients: dob, gender

6) diagnoses_icd: seq_num, icd9_code(truncated to first 3 characters)

7) procedures_icd: seq_num, icd9_code(truncated to first 3 characters)

We initially truncated the diagnoses_icd9_code to the first 3 characters, as these correspond to a "class of a disease" which was our objective, since we decided to try and predict all post-procedural complications, independent of type.

Code 4.1. SQL Code for trimming the icd9_code to first 3 digits wrt d_icd_diagnoses table

```
CREATE TABLE icd9Trunc_diagnoses AS
SELECT ROW_id, SUBSTRING(icd9_code, 1, 3) AS icd9_code,
   short_title
FROM d_icd_diagnoses;
```

The SQL code above creates the table named "icd9Trunc_diagnoses" starting from the original MIMIC-III table "d_icd_diagnoses" by selecting the variables, "Row_id" containing the joining identifier between the different tables corresponding to an individual patient, the first three characters of the variable "icd9_code" and a short_title corresponding to the "icd9_code".

As a next step, we removed all the diagnoses_icd9_code whose count is less than 30. There is no specific reason in choosing the number 30, but such a choice will ensure we focus on the more common diseases and avoid skewed results due to outliers and over-fitting.

Code 4.2. SQL Code for keeping a count of each icd9_code wrt icd9Trunc_diagnoses table

```
CREATE TABLE icd9Trunc_diagnoses_count AS
SELECT icd9_code, COUNT(icd9_code) AS total_count
FROM icd9Trunc_diagnoses
GROUP BY icd9_code
ORDER BY icd9_code DESC;
```

The script above creates a table named "icd9Trunc_diagnoses_count" starting from the previously created table "icd9Trunc_diagnoses" by selecting the variables, "icd9_code" and its corresponding count by grouping them together and ordering the icd9_code in descending order.

Code 4.3. SQL Code for removing icd9_code with less than 30 counts

```
CREATE TABLE icd9Trunc_diagnoses_gt30
AS SELECT icd9_code, total_count
FROM icd9Trunc_diagnoses_count
WHERE total_count >= 30;
```

The SQL table above named "icd9Trunc_diagnoses_gt30" is created from the previously created table "icd9Trunc_diagnoses_count" by selecting the variables, "icd9_code" whose count is greater than 30.

Code 4.4. SQL Code for linking icd9Trunc_diagnoses with icd9Trunc_diagnoses_gt30

```
CREATE TABLE icd9Trunc_diagnoses_gt30_full AS
SELECT a.ROW_id, a.icd9_code, b.icd9_code AS
```

(Code 4.4. cont'd.)

```
   icd9_code_trunc, a.short_title
FROM d_icd_diagnoses As a, icd9Trunc_diagnoses_gt30 AS b
WHERE SUBSTRING(a.icd9_code,1,3) = b.icd9_code;
```

The query above creates the table named "icd9Trunc_diagnoses_gt30_full" by linking the original MIMIC-III table "d_icd_diagnoses" with the newly created table "icd9Trunc_diagnoses_gt30" by selecting the variables, "Row_id" containing the joining identifier between the two tables corresponding to an individual patient, the "icd9_code" from the original table and the newly truncated "icd9_code" from the new table.

Code 4.5. SQL Code for linking diagnoses_icd with icd9trunc_diagnose_sgt30_full table

```
CREATE TABLE diagnoses_icd9Trunc AS
SELECT a.row_id, a.subject_id, a.hadm_id, a.seq_num, b.
   icd9_code_trunc
FROM diagnoses_icd AS a, icd9trunc_diagnoses_gt30_full as b
WHERE a.icd9_code = b.icd9_code;
```

The table "diagnoses_icd9Trunc" is joined by linking the original MIMIC-III table "diagnoses_icd" with the newly created table "icd9trunc_diagnoses_gt30_full" by selecting the variables, "Row_id", "Subject_id" and "hadm_id" containing the joining identifier between the two tables corresponding to an individual patient, the "seq_num " and the truncated "icd9_code".

The same steps are followed for procedures_icd9_code but this time we removed all the procedures_icd9_code whose count is less than 05. Again, the choice of 5 was inspired by the average number of procedures performed on a patient. Routine check-ups were excluded (few procedures) since the chance of having complications during them is extremely low.

Code 4.6. SQL Code for trimming to first 3 digits of the icd9_code wrt d_icd_procedures table

```
CREATE TABLE icd9Trunc_procedures
AS SELECT ROW_id, SUBSTRING(icd9_code, 1, 3) AS icd9_code,
   short_title
FROM d_icd_procedures;
```

The SQL code above creates the table named "icd9Trunc_procedures" starting from the original MIMIC-III table "d_icd_procedures" by selecting the variables, "Row_id" containing the joining identifier between the different tables corresponding to an individual patient, the first three characters of the variable "icd9_code" and a short_title corresponding to the "icd9_code".

Code 4.7. SQL Code for keeping a count of each icd9_code wrt icd9Trunc_procedures table

```
CREATE TABLE icd9Trunc_procedures_count
AS SELECT icd9_code, COUNT(icd9_code) AS total_count
FROM icd9Trunc_procedures
GROUP BY icd9_code
ORDER BY icd9_code DESC;
```

The above query creates the table named "icd9Trunc_procedures_count" from the previously created table "icd9Trunc_procedures" by selecting the variables, "icd9_code" and its corresponding count by grouping them together and ordering the icd9_code in descending order.

Code 4.8. SQL Code for removing icd9_code whose count is less than 05 wrt icd9Trunc_procedures_count table

```
CREATE TABLE icd9Trunc_procedures_gt05
AS SELECT icd9_code, total_count
FROM icd9Trunc_procedures_count
WHERE total_count >= 5;
```

The table named "icd9Trunc_procedures_gt05" is created from the previously created table "icd9Trunc_procedures_count" by selecting the variables, "icd9_code" whose count is greater than 05.

Code 4.9. SQL Code for linking icd9Trunc_procedures with icd9Trunc_procedures_gt05

```
CREATE TABLE icd9Trunc_procedures_gt05_full
AS SELECT a.ROW_id, a.icd9_code, b.icd9_code AS
   icd9_code_trunc, a.short_title
FROM d_icd_procedures As a, icd9Trunc_procedures_gt05 AS b
WHERE SUBSTRING(a.icd9_code,1,3) = b.icd9_code;
```

The snippet above creates the table named "icd9Trunc_procedures_gt05_full" by linking the original MIMIC-III table "d_icd_procedures" with the newly created table "icd9Trunc_procedures_gt05" by selecting the variables, "Row_id" containing the joining identifier between the two tables corresponding to an individual patient, the "icd9_code" from the original table and the newly truncated "icd9_code" from the new table.

Code 4.10. SQL Code for linking procedures_icd table with icd9trunc_procedures_gt05_full table

```
CREATE TABLE procedures_icd9Trunc
```

(Code 4.10. cont'd.)

```
AS SELECT a.row_id, a.subject_id, a.hadm_id, a.seq_num, b.
   icd9_code_trunc
FROM procedures_icd AS a, icd9trunc_procedures_gt05_full as
    b
WHERE a.icd9_code = b.icd9_code;
```

The SQL code above creates the table named "procedures_icd9Trunc" by linking the original MIMIC-III table "procedures_icd" with the newly created table "icd9trunc_procedures_gt30_full" by selecting the variables, "Row_id", "Subject_id" and "hadm_id" containing the joining identifier between the two tables corresponding to an individual patient, the "seq_num " and the truncated "icd9_code".

In the MIMIC-III database, the diagnoses are ordered with the variable seq_num starting from 1 upto seq_num of n, where n is the maximum number of diagnoses recorded for a patient against that particular admission id. For our work, we capped the value of n to be 6, since the number of patients with more than 6 diagnoses was negligible compared to the general population.

Code 4.11. SQL Code for trimming maximum seq_num

```
CREATE TABLE diagnoses_icd9Trunc_seqnum_max6 AS
SELECT *
FROM diagnoses_icd9Trunc
WHERE seq_num <= 6;
```

The table "diagnoses_icd9Trunc_seqnum_max6" is created from the previously created table "diagnoses_icd9Trunc" by selecting all the variables from "diagnoses_icd9Trunc" table whose "seq_num" is less or equal to 6.

Code 4.12. SQL Code for extracting rows with seq_num

```
CREATE TABLE diagnoses_seqnum_1 AS
SELECT row_id, subject_id, hadm_id, icd9_code_trunc AS
   icd9_code_diagnoses_1
FROM diagnoses_icd9Trunc_seqnum_max6
WHERE seq_num = 1;
```

The query above creates the table named "diagnoses_seqnum_1" from the previously created table "diagnoses_icd9Trunc_seqnum_max6" by selecting the variables, "Row_id", "Subject_id", "hadm_id" containing the joining identifier between the different tables corresponding to an individual patient, and the variable "icd9_code_trunc" containing the truncated icd9_code.

As above, five more tables are created for extracting rows with seq_num 2, 3, 4, 5 and 6 from the diagnoses_icd9Trunc_seqnum_max10 table.

A new binary column called "Complications" was created with the value 1 if any of the diagnoses_icd9_code row starting from seq_num 1 to 6 contain the icd9_code 996 (icd9_code 996 provides us the complications to certain specified Procedures). Otherwise, the result is 0. This attribute is considered to be the output variable that has to be predicted.

Code 4.13. Complications Column

```
# Appends a column for complications and adds 0 to a row
   with no 996 and 1 to a row containing 996 icd9code
Data['Compl'] = np.where((Data['icd9_code_diagnoses_1']=="
   996")|(Data['icd9_code_diagnoses_2'] == "996")|(Data['
   icd9_code_diagnoses_3'] == "996")|(Data['
   icd9_code_diagnoses_4'] == "996")|(Data['
```

(Code 4.13. cont'd.)

```
   icd9_code_diagnoses_5'] == "996")|(Data['
   icd9_code_diagnoses_6'] == "996"),1,0)
```

Once the Complications column is created, the icd9_code 996 was removed in all the diagnoses_icd9_code columns. This way, we wont be predicting the 996 occurrence, having 996 as input.

Code 4.14. 996 replaced to 0

```
# Changes all 996 into 0
Data = Data.replace("996", 0)
```

To have a consistent data set, we replaced all the diagnoses icd9_code rows starting with "V" and "E" to 0. This is a technicality of the ICD9 coding system, and we decided to explore these specific diagnoses in a future publication.

Code 4.15. Replaced rows with alphabets

```
# Replaces all the icd9codes starting with an alphabet into
    0
Data = Data.replace(["E87", "V58", "E93", "E88", "V10", "
   E93", "V45", "V49", "E91", "E94", "V15", "E81", "E84", "
   E00"],0)
```

Once the complications column is created, we retained only the icd9_code with seq_num equal to 1 and the remaining seq_num from 2 to 6 were removed. This is done because the column with seq_num equal to 1 generally gives us the primary diagnoses or the diagnoses which is more relevant to the patient's stay at hospital. Also, the count of seq_num for diagnoses computed, representing the total number of diagnoses performed for the patient

against that particular admission id. Again, we agree that removing this information may be reducing the accuracy of our model, yet for the purposes of this thesis and to make things easier to explain we decided to implement this simplification.

Code 4.16. Global replacements

```
# Collapses all Icd9 to one
for i in range(0, len(Data.index)):
    for j in range(2, 6):
        if (Data.iloc[i,j] != 0):
            Data.iloc[i,2] = Data.iloc[i,j]
            break

# Cleaning, removing column, change to numeric, rearrange
Data = Data.drop(['subject_id_x', 'hadm_id', '
   icd9_code_diagnoses_2', 'icd9_code_diagnoses_3', '
   icd9_code_diagnoses_4', 'icd9_code_diagnoses_5', '
   icd9_code_diagnoses_6', 'icd9_code_procedures_1', '
   icd9_code_procedures_2', 'icd9_code_procedures_3', '
   icd9_code_procedures_4', 'icd9_code_procedures_5', '
   icd9_code_procedures_6'], axis= 1)
```

Since the Complications column is our output variable, it was moved to the beginning of our data set to make it easy for further implementations.

Code 4.17. Re-arranging Complications column

```
# moves the Compl to first column
cols = Data.columns.tolist()
```

(Code 4.17. cont'd.)

```
cols = cols[-1:] + cols[:-1]
Data = Data[cols]
```

As the final step of pre-processing, feature scaling was performed to scale all the variables in the range of 0 and 1.

Code 4.18. Feature Scaling

```
# Making use of MinMax Scaler to scale all the variables in
    the range of 0 and 1
min_max_scaler = MinMaxScaler()
Data = min_max_scaler.fit_transform(Data)]
```

4.1.3. Variable manipulation-creation

To further analyze connections between our target variable and some characteristics of the patients, three more input variables were considered: full_los, age and procedures. The variable full_los was calculated as the difference between admittance and discharge time. Age was calculated based on the admittance time and dob. The procedures variable was calculated by taking into account the count of seq_num for procedures, representing the total number of procedures performed for the patient during that particular admission id.

All the string input attributes namely insurance, religion, martial_status, ethnicity, costcenter, first_careunit, last_careunit, prev_service and curr_service are encoded to numeric values and since we are using Neural Networks, it is always better to scale all the input attributes which will speed up all back propagation techniques.

Once all these computations, including the preliminary statistical analysis and feature selection are performed (see Section 5, the normalized dataset is exported to our local machine which serves as input to train the neural network.

4.2. Neural Network Implementation

As we mentioned earlier a Python script was written to develop and train the neural network model using the pre-processed data. We are making use of the Keras package for building and training the model while testing four different optimizers [8]: Adam, AdaGrad, RMSProp and AdaMax.

Code 4.19. List of Optimizers

```
# Defining a list of optimizers for training different
   neural network models
optimizer = ['adam', 'adagrad', 'rmsprop', 'adamax']
```

Our final data set had 4,311 data points in total. From this set, we created 100 subsets of 3,500 data points chosen in random and from each subset, 90% of the data points were used as training set and the remaining 10% were used as the test set to compute the prediction accuracy.

Code 4.20. Training and Test set split

```
# Defining the number of data points to be chosen in random
random_subset = NeuralNetworkModel.Data.sample(n = 3500)

# Defining the Input variables as X and target variable as
   y
X = random_subset.iloc[:, 1:].values
y = random_subset.iloc[:, :1].values

# Splitting the Dataset into Training set and Test set with
    the ratio 90:10
```

(Code 4.20. cont'd.)

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
   test_size = NeuralNetworkModel.alpha)
```

Our model was a 3-layer neural network with two hidden layers of 8 and 5 nodes. We came up with this final architecture after testing various possibilities as follows:

We picked a subset of 1,000 random data points from the cleaned dataset and created various architectures by varying the first hidden layer from 5 to 10 nodes and the second hidden layer from 5 to 10 nodes.

The above steps were written in a for loop repeating 100 times. The model with 8 nodes in the first hidden layer and 5 in the second hidden layer was the one with the higher average accuracy. Once the number of nodes in the first two hidden layers were obtained, a third hidden layer was created by changing the nodes from 1 to 10 and the same steps were repeated for 100 trials.

Unfortunately, the accuracy did not improve by adding the third hidden layer. Hence, the third hidden layer was removed and the two hidden layers with 8 and 5 nodes and the output layer with one node were made final. Figure 4.2 provides a better visualization on our 3-layer neural network with two hidden layers and an output layer.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 8)                 136
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 45
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 6
=================================================================
Total params: 187
Trainable params: 187
Non-trainable params: 0
```

Figure 4.2. Layer Visualization

With the architecture settled, the first algorithmic step to build a Neural network is to define the model. The sequential model was chosen from the Keras package which train the layers one after the other (linear stack of layers).

Code 4.21. Neural Network Initialization

```
# Initializing Neural Network by defining the model
model = Sequential()
```

The "Dense layer" is a command in Keras used to add a layer. Every neuron will receive an input from all the neurons of the previous layer and hence the name Dense as it is densely connected. The output_dim is used to specify the number of output nodes for each layer. init = 'uniform' is the way to define weights at each node; Here we chose random weights to be are assigned to each node in the layer using a uniform distribution. The input_dim is nothing more than the number of input nodes. input_dim should be defined only for the first layer to specify the number of input variables. In the later layers, the input variables are automatically calculated based on the output nodes of the previous layer. The activation function is applied on the sum of weighted sum and bias value to decide whether that particular node should be activated or not. We have made use of the Sigmoid function to restrict the output values between 0 and 1.

Code 4.22. Adding layers

```
# Defining the input layer and the first hidden layer
model.add(Dense(input_dim=16, activation="sigmoid", units
   =8, kernel_initializer="uniform"))

# Defining the second hidden layer
model.add(Dense(kernel_initializer="uniform", activation="
```

(Code 4.22. cont'd.)

```
    sigmoid", units=5))

# Defining the final output layer
model.add(Dense(kernel_initializer="uniform", activation="
    sigmoid", units=1))
```

In the Initialization process, the "init method" sets the first value of the weights at random. As a next step, each optimizer will apply a version of Stochastic Gradient Decent to find the best suited (optimal) set of weights in order to better predict the output. The Loss function is used to calculate the error between the truth and the prediction, and it is the one the SGD will try to minimize. We are making use of the "Sum of Squared Error" loss function here which is a typical choice for almost all predictive models. This is the summation of all the differences between actual and predicted output squared.

Code 4.23. Neural Network Compilation

```
# Compiling the Neural Network
model.compile(optimizer = opt, loss = 'mean_squared_error',
    metrics = ['accuracy'])
```

In the Optimization process, the weights are optimized in order to achieve the best model accuracy. The batch_size specifies the number of training examples in a batch after which the weights need to be updated. The argument Epoch specifies the number of iterations the algorithm will go through until it terminates. One epoch means the entire dataset completing the backward and forward propagation through the neural network once.

Code 4.24. Fitting the model

```
# Fitting the model for further prediction
model.fit(X_train, y_train, batch_size = 100, epochs =
   1000)
```

The Predicted test result is then computed and turned into a binary prediction, with value 1 corresponding to some sort of complications and 0 to no complications.

Code 4.25. Test Result Prediction

```
# Predicting the test data
y_pred = model.predict(X_test)
y_pred = y_pred > 0.5)
```

Finally, a confusion matrix is generated to explore the predictions of our model in the test set. The final accuracy is also computed.

Code 4.26. Confusion matrix and Accuracy

```
# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
accuracy_score(y_test, y_pred)
```

The above steps were part of a for loop which repeated the process 100 times on the four neural networks each for one optimizer. This created 100 accuracy scores for each model (400 accuracy scores in total). The accuracy scores were saved in a .csv file for further analysis.

Chapter 5.
Experimental Results

5.1. Basic Statistics

In this section, we present some of the basic numerical descriptors of our dataset and the results of our initial analysis. After the pre-processing stage, our clean dataset contained a total of 4311 datapoints.

The histogram in Figure 5.1 shows the count of procedures performed on each patient with x-axis being the number of procedures performed and y-axis being the number of patients.

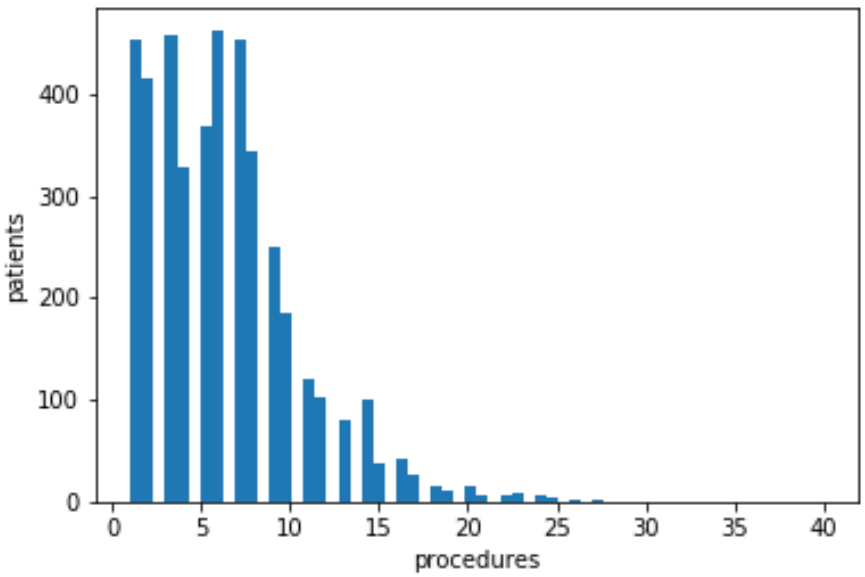


Figure 5.1. Histogram of the count of procedures performed for the patients

As we can see, the majority of patients had up to 10 procedures performed on them with an average of 6.32 procedures. As we see in table 5.1, $\frac{1166}{4311}$ or about 27% of the patients suffered from some form of complication during their hospital stay. This will be used to calculate the "No Information Rate" that our prediction model will need to outperform.

The histogram in Figure 5.2 shows the count of diagnoses performed on each patient with x-axis being the number of diagnoses performed and y-axis being the number of

Table 5.1. Patients with 996 and no 996

	No	**Yes**
Complications	3145	1166

patients.

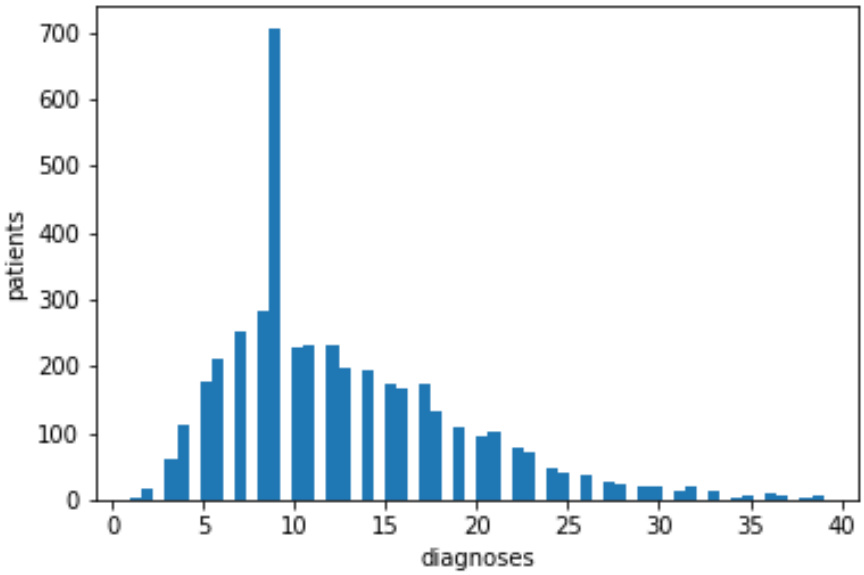


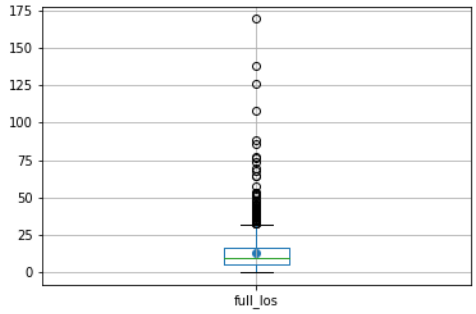
Figure 5.2. Histogram of the count of diagnoses performed for the patients

As we can see, the majority of patients had up to 10 diagnoses, with an average of 12.73 diagnoses.

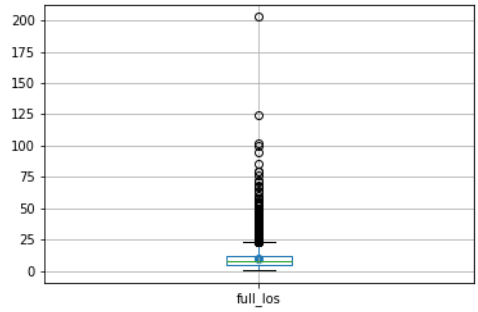
The Box-plots in Figure 5.3 show the full_los of the patients with the icd9 code 996 versus the full_los of the patients with no icd9 code 996. As we see in Figure 5.3, $\frac{11538}{876}$ or about 13% of the patients had their full_los with the icd9 code 996.

The Box-plots in Figure 5.4 show the full length of stay of the patients with icd9 code 996 versus the full_los of the patients with no icd9 code 996 if the full_los is capped to 50 days(removing extreme outliers). As we see in Figure 5.4, $\frac{12161}{844}$ or about 15% of the patients had their full_los with the icd9 code 996 if the full_los is capped to 50 days.

It is obvious that the length of stay is on average longer if one has hospital complications as expected.

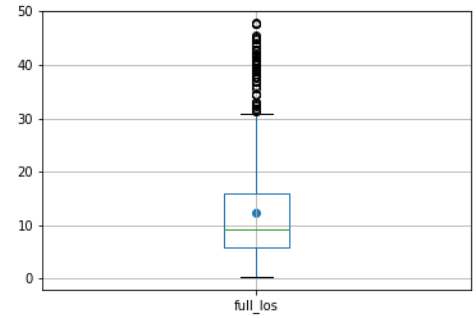


(a) Box-plot for full_los with icd9 code 996

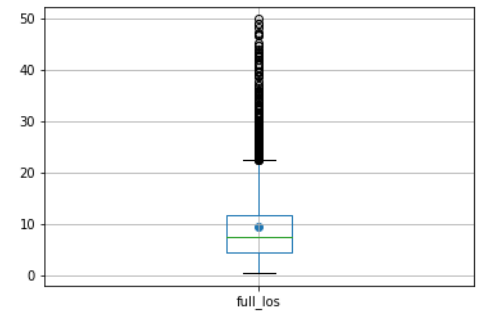


(b) Box-plot for full_los with no icd9 code 996

Figure 5.3. Box-plot for full_los



(a) Box-plot for full_los with icd9 code 996



(b) Box-plot for full_los with no icd9 code 996

Figure 5.4. Box-plot for full_los capped to 50 days

The Histogram in Figure 5.5 depicts the age of all patients in our dataset. The majority of them were above 50 years with an average age of 62.69. All our patients' ages are capped to 89 years. Patients who were older than 89 years in the database have had their date of birth shifted to obscure their age to comply with HIPAA regulations.

The majority of patients in the cleaned dataset were male as one can see from table 5.2. One of our future plans is to analyze those two populations separately and compare the performance with the initial model.

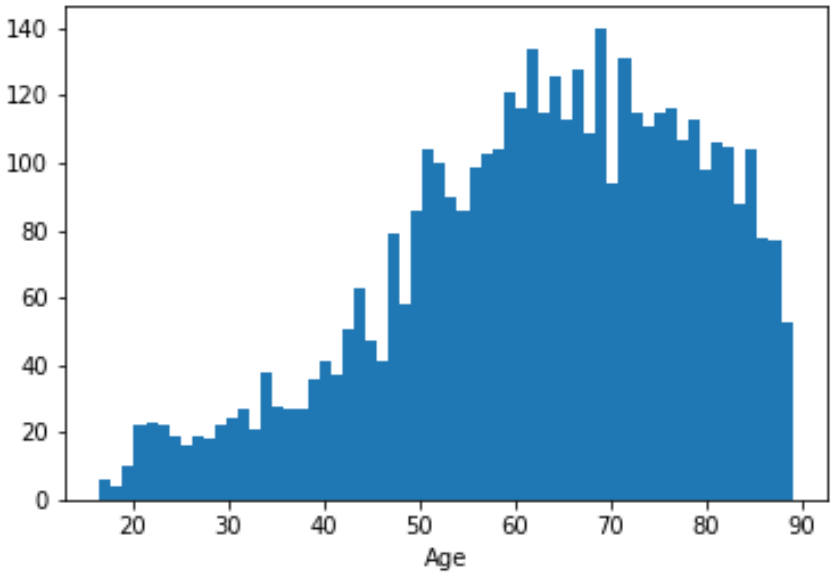


Figure 5.5. Histogram of all the patients age

Table 5.2. Count of Male vs Female

	Male	**Female**
Age	2636	1675

5.2. Feature Selection

Most data analytics efforts start with some sort of "feature selection" to identify the key input variables that will be used in the prediction model. Neural Networks don't really need this step since their architecture has the ability to tone down irrelevant information, yet it is always a good practice to check what the right input variables are at the onset.

A contingency table summarizes the relationship between categorical variables in terms of frequency counts. A chi-square test is conducted on the contingency table to test for independence between the categorical variables involved. A value less than 0.05 implies both the variables are dependent, following the classic hypothesis test work-flow.

5.2.1. Age vs Complications

Code 5.1. Chi-square Analysis for Age

```
# Calculates the median value of Age
Age_median = Data['Age'].median()

# Appends a column for categorical value of Age. If the age
    is greater than the median, AgeC is changed to 1. else
   0
Data['AgeC'] = np.where(Data['Age']>=Age_median,1,0)

# Table of AgeC vs. Compl
Cont_Age_Compl = pd.crosstab(index=Data['AgeC'], columns=
   Data['Compl'])
Cont_Age_Compl.index= [0, 1]

# Finds the chi-square value for Age
chi2, Cont_Age_Compl_p, Cont_Age_Compl_dof,
   Cont_Age_Compl_expected = sc.chi2_contingency(
   Cont_Age_Compl.values)
```

The following contingency table shows the relationship between the two categorical variables Age and Complications, Age being the rows and Complications being the columns. We discretized age by creating two groups below and above the median age.

Based on the chi-square test, the p-value for the above contingency table is **0.005327**. This value is much smaller than **0.05**. This provides enough evidence to claim that the categorical variables Age and Complications are dependent. The expected frequency counts

Table 5.3. Contingency table values for Age vs Complications

	No	Yes	Total
Young	1531	624	2155
Old	1614	542	2156
Total	3145	1166	4311

for the two categorical variables was as follows:

Table 5.4. Expected contingency table values for Age vs Complications

	No	Yes	Total
Young	1572.135	582.864	2154.999
Old	1572.864	583.135	2155.999
Total	3144.999	1165.999	4310.998

5.2.2. Full LOS vs Complications

Code 5.2. Chi-square Analysis for full_los

```
# Calculates the median value of full_los
full_los_median = Data['full_los'].median()

# Appends a column for categorical value of full_los. If
   the full_los is greater than the median, full_losC is
   changed to 1. else 0
Data['full_losC'] = np.where(Data['full_los']>=
   full_los_median,1,0)

# Table of full_losC vs. Compl
Cont_LOS_Compl = pd.crosstab(index=Data['full_losC'],
   columns=Data['Compl'])
Cont_LOS_Compl.index= [0, 1]
```

(Code 5.2. cont'd.)

```
# Finds the chi-square value for full_los
chi2, Cont_LOS_Compl_p, Cont_LOS_Compl_dof,
   Cont_LOS_Compl_expected = sc.chi2_contingency(
   Cont_LOS_Compl.values)
```

The following contingency table shows the relationship between the variables full_los and Complications, full_los being the rows and Complications being the columns.

Table 5.5. Contingency table values for Full_los vs Complications

	No	Yes	Total
Short	1674	481	2155
Long	1471	685	2156
Total	3145	1166	4311

Based on the chi-square test, the p-value for the above contingency table is **3.6274e-12** which is way smaller than **0.05**. It is obvious that the categorical variables full_los and Complications are dependent on each other. T The expected frequency counts for the two categorical variables was as follows:

Table 5.6. Expected contingency table values for full_los vs Complications

	No	Yes	Total
Short	1572.135	582.864	2154.999
Long	1572.864	583.135	2155.999
Total	3144.999	1165.999	4310.998

5.2.3. Procedures vs Complications

Code 5.3. Chi-square Analysis for procedures

```
# Calculates the median value of procedures
proc_median = Data['procedures'].median()

# Appends a column for categorical value of procedures. If
   the procedures is greater than the median, procC is
   changed to 1. else 0
Data['procC'] = np.where(Data['procedures']>=proc_median
   ,1,0)

# Table of procC vs. Compl
Cont_proc_Compl = pd.crosstab(index=Data['procC'], columns=
   Data['Compl'])
Cont_proc_Compl.index= [0, 1]

# Finds the chi-square value for procedures
chi2, Cont_proc_Compl_p, Cont_proc_Compl_dof,
   Cont_proc_Compl_expected = sc.chi2_contingency(
   Cont_proc_Compl.values)
```

The following contingency table shows the relationship between two categorical variables Procedures and Complications, Procedures being the rows and Complications being the columns.

Based on the chi-square test,the p-value for the above contingency table is **5.006e-07**. This value is much smaller than 0.05 which provides enough evidence to claim that the

Table 5.7. Contingency table values for Procedures vs Complications

	No	Yes	Total
Few	1400	620	2020
Many	1745	546	2291
Total	3145	1166	4311

categorical variables Procedures and Complications are dependent on each other.

After performing a chi-square test on the above contingency table, the expected frequency counts for the two categorical variables to be independent of each other are as follows:

Table 5.8. Expected contingency table values for Procedures vs Complications

	No	Yes	Total
Few	1473.648	546.351	2019.999
Many	1671.351	619.648	2290.999
Total	3144.999	1165.999	4310.998

5.2.4. Gender vs Complications

Code 5.4. Chi-square Analysis for gender

```
# Calculates the median value of gender
gender_median = Data['gender'].median()

# Appends a column for categorical value of gender. If the
   gender is greater than the median, genderC is changed to
    1. else 0
Data['genderC'] = np.where(Data['gender']>=gender_median
   ,1,0)

# Table of genderC vs. Compl
```

(Code 5.4. cont'd.)

```
Cont_gender_Compl = pd.crosstab(index=Data['genderC'],
   columns=Data['Compl'])
Cont_gender_Compl.index= [0, 1]


# Finds the chi-square value for gender
chi2, Cont_gender_Compl_p, Cont_gender_Compl_dof,
   Cont_gender_Compl_expected = sc.chi2_contingency(
   Cont_gender_Compl.values)
```

The following contingency table shows the relationship between Gender and Complications, Gender being the rows and Complications being the columns.

Table 5.9. Contingency table values for Gender vs Complications

	No	Yes	Total
Female	1196	479	1675
Male	1949	687	2636
Total	3145	1166	4311

Based on the chi-square test, the p-value for the above contingency table is **5.006e-07**. This value is much smaller than 0.05. So, this provides enough evidence to claim that the categorical variables Gender and Complications are dependent on each other.

Table 5.10. Expected contingency table values for Gender vs Complications

	No	Yes	Total
Female	1473.648	546.351	2019.999
Male	1671.351	619.648	2290.999
Total	3144.999	1165.999	4310.998

Considering the above four chi-square values, all four input variables: Age, full_los, procedures and gender seem to be connected to complications and hence these input variables

were definitely included in the prediction dataset. But, since we choose the non-parametric method of Neural Networks, we decided to include all the input variables. In the future we will explore further those relationships and try to determine if they are indeed factors that influence complication or some sort of co-dependent variables on another hidden one.

5.3. Results

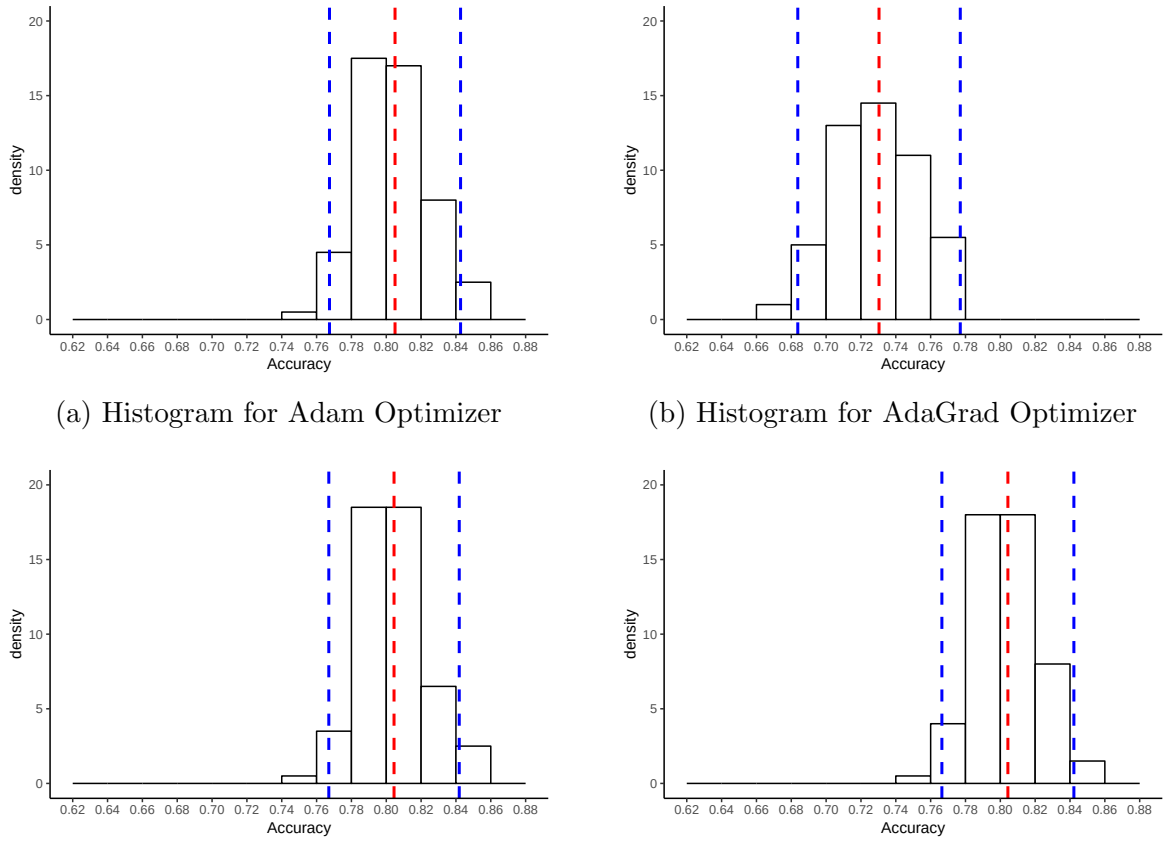
The Accuracy table 5.11 below shows the average accuracy of the four different optimizers: adam, adagrad, rmsprop and adamax. Each accuracy score is the average of a 100 runs on randomly chosen subsets common for all optimizers each time. The Neural Networks built using these optimizers: adam, rmsprop and adamax produced models that are approximately 80% accurate in predicting the post-procedural complications.

To be specific, the neural network model built using the adam optimizer had an average accuracy score of 80.5% with an average processing time of 133.68 micro seconds. The rmsprop optimizer had an average accuracy score of 80.4% with an average processing time of 127.78 micro seconds. The adamax optimizer had an average accuracy score of 80.4% with an average processing time of 132.27 micro seconds except for the adagrad optimizer which had an average accuracy score of 73% with an average processing time of 126.06 micro seconds. The rmsprop optimizer had a better processing time when compared with the adam and adamax optimizers.

Table 5.11. Accuracy Table

Average_Accuracy	**Optimizer**	**Processing_time**
80.50%	adam	133.68
73.03%	adagrad	126.06
80.44%	rmsprop	127.78
80.43%	adamax	132.27

In comparison with the neural network model, the multi-linear model had an accuracy score of 73.84%. This proves that the neural network model outperformed the multi-linear model.



(a) Histogram for Adam Optimizer

(b) Histogram for AdaGrad Optimizer

(c) Histogram for RMSProp Optimizer

(d) Histogram for AdaMax Optimizer

Figure 5.6. Basic Histograms of the Accuracy for the four different optimizers

Furthermore, Figure 5.6 pictorially represents the Accuracy scores of all the four optimizers: Adam, AdaGrad, RMSProp and AdaMax in the form of histograms. RMSProp has a better accuracy and is much faster when compared with the other three optimizers: Adam, AdaGrad and AdaMax. Though the Adam Optimizer has a higher accuracy when compared with the RMSProp (0.06% more), it is slower than the RMSProp optimizer. When the data is scaled up, the accuracy doesn't vary much but the processing time will be substantially different, with the Adam Optimizer being much slower than the RMSProp. Considering all these factors, RMSProp is probably the one best suited for our prediction algorithm.

The Accuracy table is represented in the form of a scatter plot as shown in Figure 5.7. The x-axis is the Processing Time and the y-axis is the Average Accuracy. The red dotted

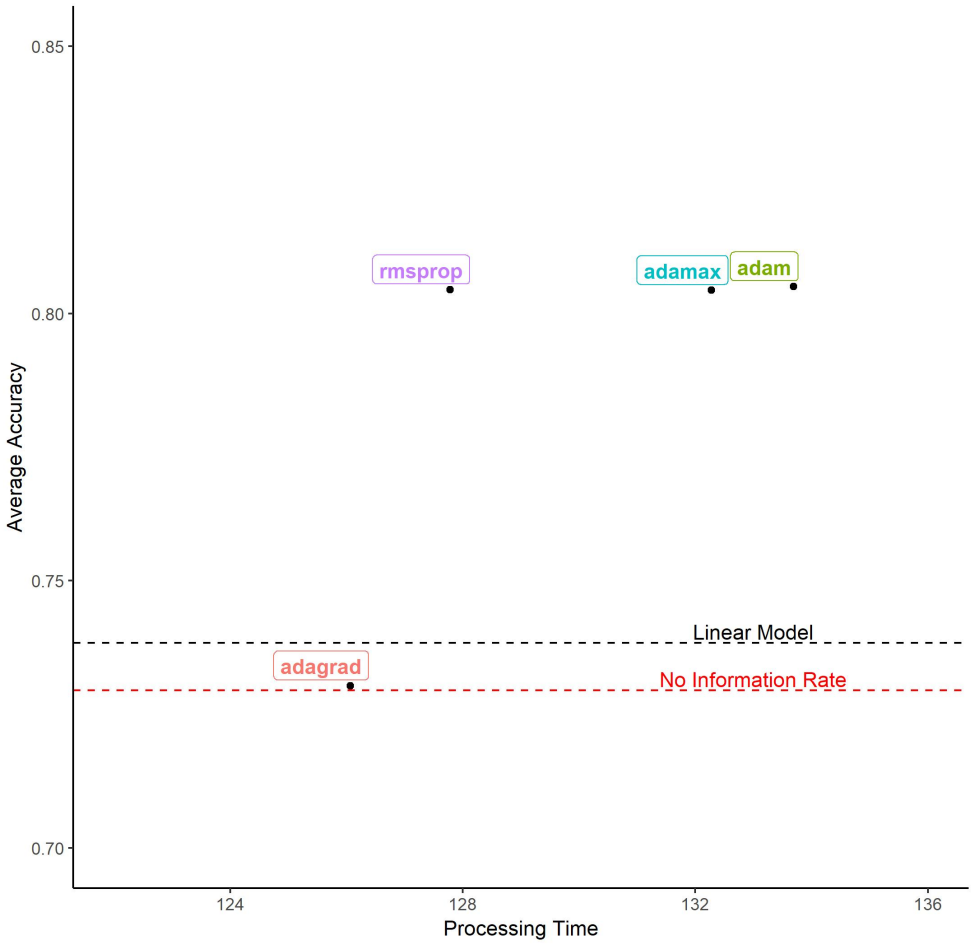


Figure 5.7. Scatter plot for the four optimizers

line represents the No Information Rate (NIR) of 72.95%. The No Information Rate (NIR) acts as the base line representing the biggest class which is class 0(no complications) in our case. Our output variable(Complications) is binary with classes 0 (no complications) and 1(with complications). The horizontal black dotted line represents the linear model with an accuracy of 73.84%. When compared to the NIR, linear model just doesn't work and also the AdaGrad optimizer works poorly too. RMSProp provides the best processing time (127.78 micro seconds) for its accuracy (80.44%) when compared with the other optimizers and also beats the NIR by at least 7%. This is a non-negligible prediction improvement especially for the medical field and if one considers the general purpose of the model. We believe that a more specialized model will increase the accuracy scores even more.

Chapter 6.
Conclusion-Future Work

In this thesis, we have investigated the use of neural networks in predicting complications when certain specified procedures are performed during a patient's hospital stay. Various characteristics of the patient as well as specific details of the patient's visit to the hospital at that point in time were used to make the prediction. Predicting such post-procedural complications provides better support for clinicians, hospital administrators and patients.

The model was trained using part of the MIMIC-III database and four different SGD optimizers Adam, AdaGrad, RMSProp and AdaMax on a Neural Network architecture. Almost all, optimizers provided similar results which were stable both in terms of processing time and accuracy. Our best model outperformed the linear model in terms of prediction accuracy (80% vs 73%) which was also almost the NIR .

To conclude, this thesis provides a post-procedural complications prediction system based on machine learning, an algorithm that has has not been reported till date.

All codes are available in our github repository found here https://github.com/namrathamohan/Post-Procedural-Complications.

The dataset can be downloaded from the MIMIC-III website upon following certain steps and completing the HIPAA certification. As a part of future work, an analysis based on the activation functions is to be done. We performed a small part of it by replacing the activation function in the first two layers with the "relu" activation function.

The Accuracy table 6.1 below shows the average accuracy of the four different optimizers: adam, adagrad, rmsprop and adamax by using the "relu" activation function in the first two layers and "Sigmoid" in the third layer. We noticed a slight improvement in processing time and accuracy as one can see in figure 6.1. Thus our first priority would be to explore other possible architectures like this one.

Another option is to repeat our analysis for a larger number of data points is to be

Table 6.1. Accuracy Table for relu

Average_Accuracy	Optimizer	Processing_time
81.01%	adam	130.11
78.97%	adagrad	121.54
81.95%	rmsprop	123.85
81.30%	adamax	127.87

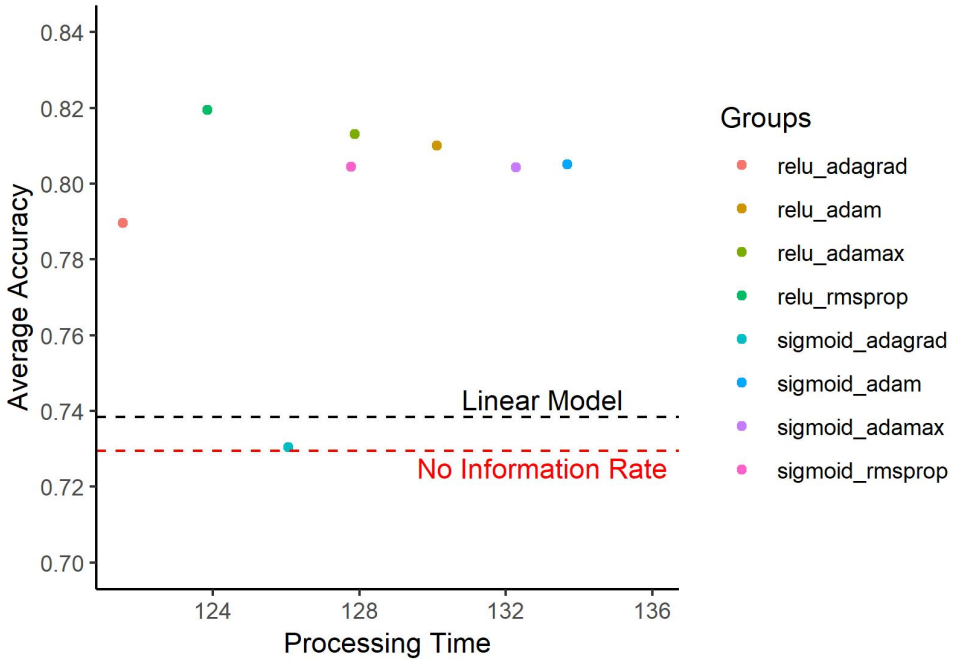


Figure 6.1. Scatter plot for the four optimizers with relu activation function

done. MIMIC is growing every year with new data points added to it periodically. As the number of data points is scaled up, it would be interesting to monitor the performance of the different optimizers and see if it will change or remain the same.

Also, an interesting future venue would be to utilize the trained algorithms on a completely different dataset if that becomes available in the future, and see if the algorithm is transferable and scalable, or identify and perform necessary corrections for it to be so.

Further development also includes the comparison of our model with another predictive algorithm using support vector machines on the same dataset. We are confident that the NN's will outperform SVM's but the trade-off in time might be worth it, since SVM's are

orders of magnitude faster thatn NN's on the same number of datapoints.

Finally, we should note here that while this thesis was finalized, the new standard (ICD10) came in to affect. Thus our code will need a slight update so that similar results can be obtained based on the available translation table between the two standards. We anticipate this to require minimal updates only on the pre-processing step.

References

[1] *Postgresql database*, Portions copyright 1996-2018, the postgresql global development group portions copyright 1994, the regents of the university of california permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without a written agreement is hereby granted, provided that the above copyright notice and this paragraph and the following two paragraphs appear in all copies. in no event shall the university of california be liable to any party for direct, indirect, special, incidental, or consequential damages, including lost profits, arising out of the use of this software and its documentation, even if the university of california has been advised of the possibility of such damage. the university of california specifically disclaims any warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. the software provided hereunder is on an "as is" basis, and the university of california has no obligations to provide maintenance, support, updates, enhancements, or modifications. https://www.postgresql.org.

[2] *Python programming language*, 2001. https://www.python.org.

[3] *Numpy package*, 2005. www.numpy.org/.

[4] *Pandas: powerful python data analysis toolkit*, 2011. https://pandas.pydata.org/pandas-docs/stable/.

[5] *Icd9 diagnosis codes*, 2012. www.icd9data.com/2012/Volume1/800-999/996-999/996/default.htm.

[6] *Artificial neural networks in healthcare: A short review* (2013). http://www.openclinical.org/neuralnetworksinhealthcare.html.

[7] *Keras: Usage of activations* (2015). https://keras.io/activations/.

[8] *Keras: Usage of optimizers*, 2015. https://keras.io/optimizers/.

[9] *What is health informatics*, USF Health (2018). https://www.usfhealthonline.com/resources/key-concepts/what-is-health-informatics/.

[10] Martin Abadi et al., *Tensorflow: Large-scale machine learning on heterogeneous systems* (2015). http://tensorflow.org/.

[11] Rami Al-Rfou et al., *Theano: A python framework for fast computation of mathematical expressions*, arXiv e-prints (2016). http://arxiv.org/abs/1605.02688.

[12] Sidath Asiri, *Machine learning classifiers* (2018). https://towardsdatascience.com/machine-learning-classifiers-a5cc4e1b0623.

[13] Jason Browniee, *Gentle introduction to the adam optimization algorithm for deep learning* (2017). https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

[14] Budde et al., *A prognostic computer model to individually predict post-procedural complications in interventional cardiology; the intervent project*, European heart journal **20** (1999), no. 5, 354–363.

[15] François Chollet et al., *Keras: The python deep learning library*, 2015. https://keras.io.

[16] Caroline Clabaugh et al., *The intellectual excitement of computer science.* (2000). https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/index.html.

[17] Wikipedia contributor, *Stochastic gradient descent* (2018). https://en.wikipedia.org/wiki/Stochastic_gradient_descent.

[18] Wikipedia contributors, *Artificial neural network — Wikipedia, the free encyclopedia* (2018). https://en.wikipedia.org/wiki/Artificial_neural_network.

[19] ———, *Python (programming language)* (2018). https://en.wikipedia.org/wiki/Python_(programming_language).

[20] Thomas Desautels et al., *Prediction of sepsis in the intensive care unit with minimal electronic health record data: A machine learning approach*, JMIR Med Inform **4** (2016).

[21] Niklas Donges, *Gradient descent in a nutshell* (2018). https://towardsdatascience.com/gradient-descent-in-a-nutshell-eaf8c18212f0.

[22] John Duchi, Elad Hazan, and Yoram Singer, *Adaptive subgradient methods for online learning and stochastic optimization*, July 2011, pp. 2121–2159. http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf.

[23] Rohith Gandhi, *A look at gradient descent and rmsprop optimizers* (2018). https://towardsdatascience.com/a-look-at-gradient-descent-and-rmsprop-optimizers-f77d483ef08b.

[24] Thanos Gentimis et al., *Predicting hospital length of stay using neural networks on mimic iii data*, 2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech) (2017), 1194–1201.

[25] gk, *How neural networks work* (2017). https://chatbotslife.com/how-neural-networks-work-ff4c7ad371f7.

[26] Geoffrey Hinton et al., *Neural networks for machine learning*. http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[27] Anil K. Jain, Anil K. Jain, Jianchang Mao, Robert P.W. Duin, and Jianchang Mao, *Statistical pattern recognition: A review* (2000). http://www4.comp.polyu.edu.hk/~csajaykr/myhome/teaching/biometrics/spr_pami.pdf.

[28] Alistair E. W. Johnson, Tom J. Pollard, Lu Shen, Li-wei H. Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo A. Celi, and Roger G. Mark, *Mimic-iii, a freely accessible critical care database*, Scientific Data **3** (May 2016), 160035+. https://mimic.physionet.org/about/mimic/.

[29] Diederik P. Kingma and Jimmy Ba, *Adam: A method for stochastic optimization*, 2014. https://arxiv.org/pdf/1412.6980.pdf.

[30] Emma Lindberg, *Light-up neuron* (2017). https://www.brainfacts.org/For-Educators/For-the-Classroom/2017/Light-Up-Neuron-092717.

[31] Bernard Marr, *What are artificial neural networks - a simple explanation for absolutely anyone* (2018).

[32] Shubham Panchal, *Artificial neural network - mapping the human brain*, 2018. https://medium.com/predict/artificial-neural-networks-mapping-the-human-brain-2e0bd4a93160.

[33] The pgAdmin Development Team, *pgAdmin an open source administration and development platform for PostgreSQL*. https://www.pgadmin.org.

[34] John Sullivan, *6 steps to write any machine learning algorithm* (2018). https://towardsdatascience.com/6-steps-to-write-any-machine-learning-algorithm-from-scratch-perceptron-case-study-335f638a70f3.

[35] Wikipedia contributors, *Amoeba (operating system) — Wikipedia, the free encyclopedia* (2018). https://en.wikipedia.org/w/index.php?title=Amoeba_(operating_system)&oldid=847604296.

Vita

Namratha Mohan is from Bangalore, India. She obtained her Bachelor's degree in Electronics and Communication Engineering in 2011 from City Engineering College, Bangalore, affiliated to the Visveswaraya Technological University, Karnataka, India. She then worked for about four years as a System Engineer with IBM India Pvt. Ltd. In Spring 2017, she began her masters program in the Department of Computer Science at Louisiana State University. Her primary interests include Database Systems and Machine Learning.