Prediction of Early Chronic Kidney Disease of Unknown Etiology (CKDu)

Final Report

Under the guidance of Dr. Anasuya Ganguly

Prem Gupta	2017A7PS0063G
Somil Gupta	2017A7PS0142G
Vinit Kumar Munjal	2017A7PS0148G

Introduction

Chronic Kidney disease (CKD) is a global health problem and the 12th leading cause of death worldwide (Jha et al., 2013) with common risk-factors being diabetes and hypertension (Vassalotti et al., 2016). However, recently, a growing concern has aroused about a new-form of CKD not caused by these risk-factors termed as CKD of unknown-etiology (CKDu) (Jayatilake et al., 2013). It is prominent in developing-countries like Sri-Lanka (Wanigasuriya, 2014), India (Singh et al., 2013)and some central-American countries (Costa-Rica, Nicaragua, Panama, and El-Salvador) (CorreaRotter et al., 2014). This disease affects the kidney's tubular interstitium belonging to the type-chronic tubulointerstitial nephritis (CTN) (Mackensen and Billing, 2009), caused by chronic exposure to environmental toxins like heavy metals (lead, arsenic, cadmium), mycotoxins, pesticides (diazinon) (Weaver et al., 2015). Heavy-metals cause nephrotoxicity at high-exposure levels, but it can cause toxicity even at low-levels on chronic exposure due to bio-accumulation and decreased clearance rate from the kidneys (Ferraro et al., 2010).

To correctly estimate the extent of the spread of disease and understand the environmental risk factors underlying CKDu-etiology, biochemical analyses of CKDu affected and non-affected individual’s blood and detailed hydro-geochemical analyses of CKDu-affected and non-affected region's groundwater (drinking-water) has to be conducted. The measurements include information of the individuals like their height, weight, blood pressure, and information about the groundwater sample like the pH, total alkalinity, hardness, amount of sodium, potassium present in it and trace metals profile of the groundwater.

Through this project, we are using Machine Learning and Artificial Neural Networks (ANNs) to predict and classify patients as having CKD of unknown-etiology (ckdu) or not. In the early stages of chronic kidney disease, an individual may have few signs or symptoms. Chronic kidney disease may not become apparent until the kidney function of the person is significantly impaired. With the help of Machine Learning, we can predict the disease in early stages on the basis of past data of other cases and help in further treatment of patients. To achieve this, the most important part is the data collection of previous patients affected by the disease. We use the dataset available in the UCL ML repository for creating the first ML model. We plan to test our model on a local dataset to improve the accuracy.

METHODOLOGY

Dataset:

We use the Chronic_Kidney_Disease Data Set that is provided by UCL Machine Learning Repository for open use. The data was taken over a 2-month period in India with 25 features (eg, red blood cell count, white blood cell count, etc). The target is the 'classification', which is either 'ckd' or 'notckd' - ckd=chronic kidney disease. There are 400 rows. The data needs cleaning: in that it has NaNs and the numeric features need to be forced to float. Basically, we were instructed to get rid of ALL ROWS with Nans, with no threshold - meaning, any row that has even one NaN, gets deleted.

Programming:

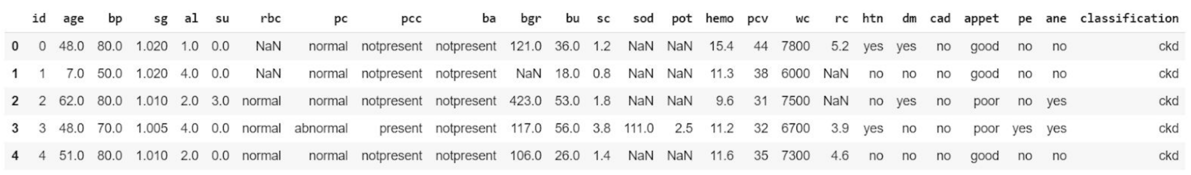
We are going to use Python language to write the code for this project. The first part will be to import libraries so that we can use the pre-existing functions available in them. This makes our code clean, fast and optimized.

Loading the dataset:

```
#Import Libraries
import glob
from keras.models import Sequential, load_model
import numpy as np
import pandas as pd
import keras as k
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
import matplotlib.pyplot as plt
```

Next, we load the dataset that we are going to use.

```
#load the data
df = pd.read_csv("../input/kidney_disease.csv")
#Print the first 5 rows
df.head()
```



	id	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc	sod	pot	hemo	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	121.0	36.0	1.2	NaN	NaN	15.4	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	NaN	18.0	0.8	NaN	NaN	11.3	38	6000	NaN	no	no	no	good	no	no	ckd
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	423.0	53.0	1.8	NaN	NaN	9.6	31	7500	NaN	no	yes	no	poor	no	yes	ckd
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.0	56.0	3.8	111.0	2.5	11.2	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	106.0	26.0	1.4	NaN	NaN	11.6	35	7300	4.6	no	no	no	good	no	no	ckd

(first 5 rows of the dataset)

Each row represents a patient and the classification whether he is prone to ckd or not.

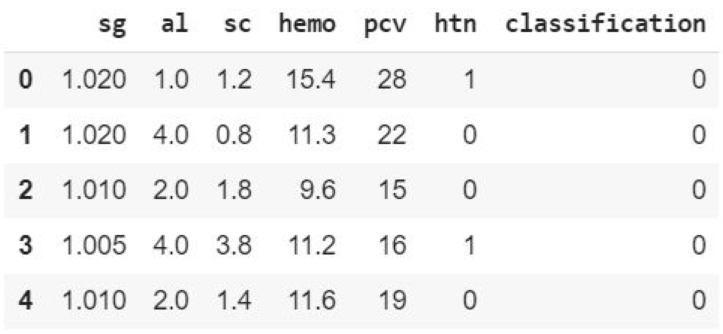
Data Manipulation: Clean the data

As the data might contain some rows with missing entry or invalid data, we need to clean the data before proceeding further. Also, there are many columns so we will have to choose to retain only some of them and drop others.

```
#Create a list of columns to retain
columns_to_retain = ["sg", "al", "sc", "hemo",
                         "pcv", "wbcc", "rbcc", "htn", "classification"]

#columns_to_retain = df.columns, Drop the columns that are not in
columns_to_retain
df = df.drop([col for col in df.columns if not col in columns_to_retain],
axis=1)

# Drop the rows with na or missing values
df = df.dropna(axis=0)
```

```
#Transform non-numeric columns into numerical columns
for column in df.columns:
        if df[column].dtype == np.number:
            continue
        df[column] = LabelEncoder().fit_transform(df[column])
```

```
#Print / show the first 5 rows of the new cleaned data set
df.head()
```



	sg	al	sc	hemo	pcv	htn	classification
0	1.020	1.0	1.2	15.4	28	1	0
1	1.020	4.0	0.8	11.3	22	0	0
2	1.010	2.0	1.8	9.6	15	0	0
3	1.005	4.0	3.8	11.2	16	1	0
4	1.010	2.0	1.4	11.6	19	0	0

<u>Data Manipulation: Split & Scale the data</u>

Now we will split the data set into an independent data set that we will call X which is the feature data set and a dependent data set that we will call y which is the target data set. We do this to ensure that we can test the accuracy of the model after training it.

```python
#Split the data into independent'X'(the features) and dependent 'y' variables
(the target)
X = df.drop(["classification"], axis=1)
y = df["classification"]
```

```python
#Feature Scaling
#the min-max scaler method scales the dataset so that all the input features
lie between 0 and 1 inclusive
x_scaler = MinMaxScaler()
x_scaler.fit(X)
column_names = X.columns
X[column_names] = x_scaler.transform(X)
```

```python
#Split the data into 80% training and 20% testing & Shuffle the data before
splitting
X_train,  X_test, y_train, y_test = train_test_split(
        X, y, test_size= 0.2, shuffle=True)
```

We have scaled the dataset so that all features lie between 0 and 1 inclusive. Next, we keep 20% of the data to reserve for the testing model.

<u>Build The Model (Artificial Neural Network):</u>

Now, we will create our ANN. We will take the help of Keras library to implement this. First, we create the model's architecture, then we add 2 layers, the first layer with 256 neurons and the ‘ReLu’ activation function with a normal distribution initializer for the weights. Since that layer is the first layer we specify the number of features/columns in the data set len(X.columns). The second layer (also the last layer), has 1 neuron and use the ‘hard_sigmoid’ activation function.

```
#Build the model
model = Sequential()
model.add(Dense(256, input_dim=len(X.columns),
                    kernel_initializer=k.initializers.random_normal(seed=13),
activation="relu"))
model.add(Dense(1, activation="hard_sigmoid"))
```

We compile the model, and give it the loss function called ‘binary_crossentropy’ which is a loss function used for binary classification, it measures how well the model did on training and then tries to improve on it using the optimizer.
The optimizer that we are using is called the ‘adam’ optimizer. We will get some metrics on the model's accuracy to see how well it does.

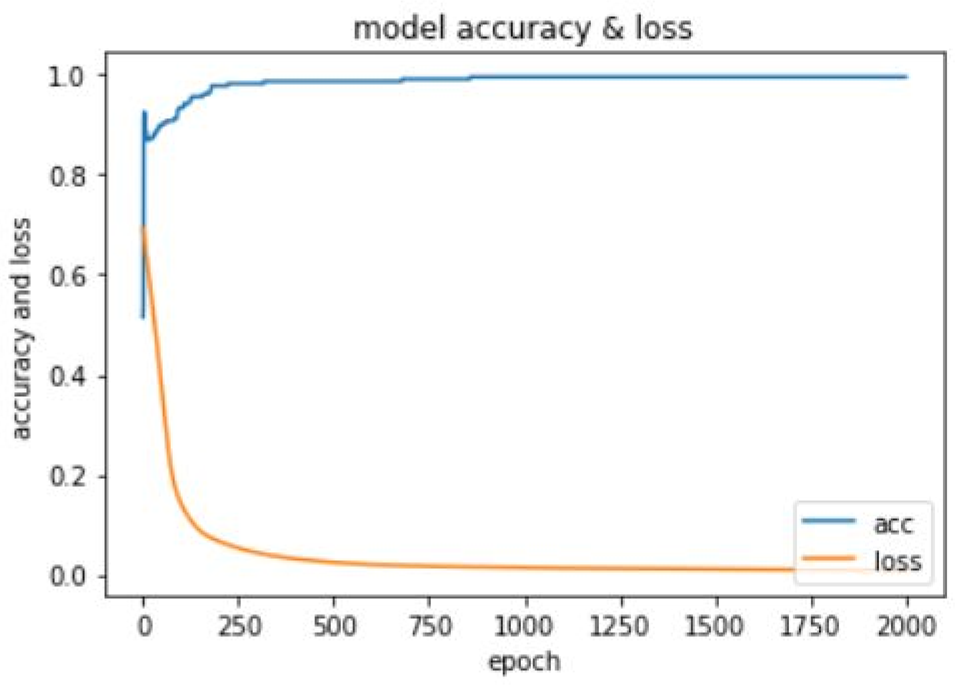
```
#Compile the model
# Loss measuers how well the model did on training , and then tries to improve
on it using the optimizer.
# The loss function we will use is binary_crossentropy for binary (2) classes.
model.compile(loss='binary_crossentropy',
                  optimizer='adam', metrics=['accuracy'])

#Train the model
history = model.fit(X_train, y_train,
                    epochs=2000, #The number of iterations over the entire
dataset to train on
                    batch_size=X_train.shape[0]) #The number of samples per
gradient update for training
```

Now we save our newly created model.

```
#Save the model
model.save("ckd.model")
```

We plot the accuracy and loss of the model to see how well it performs using the matplotlib library.

```
#Visualize the models accuracy and loss
plt.plot(history.history["acc"])
plt.plot(history.history["loss"])
plt.title("model accuracy & loss")
plt.ylabel("accuracy and loss")
plt.xlabel("epoch")
plt.legend(['acc', 'loss'], loc='lower right')
plt.show()
```



Further Work:

Our current model used the dataset that was available in the UCL ML repository. We plan to test it on a real dataset collected from the districts of Goa with patients having CKD symptoms and diseases. We can then improve the accuracy of the model further. We might have to change the columns and the model’s parameters to fit it best into the latest dataset.