

**NEAR EAST UNIVERSITY**

## **FACULTY OF ENGINEERING**

## **COMPUTER ENGINEERING DEPARTMENT**

## **AIE511 ARTIFICIAL NEURAL NETWORKS**

**ASSIGNMENT III**

**DILIGENCE VAGERE**

**20206993**

**SPRING 2020/2021**

## Onset Diabetes Predictions in Patients

This project was created to predict whether patients with provided medical records data would have onset of diabetes within the next five years. This system will return either 1 or 0 with the former indicating that the patient is predicted to have onset diabetes while the latter indicates that the patient is predicted to not have onset diabetes. This in turn makes our system a binary classification problem.

The system has 8 numerical input variables which can be used to determine the likelihood of onset diabetes, namely:

* Number of times pregnant
* Plasma glucose concentration a 2 hours in an oral glucose tolerance test
* Diastolic blood pressure (mm Hg)
* Triceps skin fold thickness (mm)
* 2-Hour serum insulin (mu U/ml)
* Body mass index (weight in kg/(height in m)^2)
* Diabetes pedigree function
* Age (years)

The system has one output variable which is the class variable with a value of either 0 or 1.

The system was implemented using Python3. The model was defined by the Keras package and then compiled using the Theano package. After the model is created, the system writes the model to a .h5 file and this model can then be loaded and used in subsequent predictions or evaluation

Information on the training dataset

URL: <https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv>

1. Title: Pima Indians Diabetes Database

2. Sources:

(a) Original owners: National Institute of Diabetes and Digestive and

Kidney Diseases

(b) Donor of database: Vincent Sigillito (vgs@aplcen.apl.jhu.edu)

Research Center, RMI Group Leader

Applied Physics Laboratory

The Johns Hopkins University

Johns Hopkins Road

Laurel, MD 20707

(301) 953-6231

(c) Date received: 9 May 1990

3. Past Usage:

1. Smith,~J.~W., Everhart,~J.~E., Dickson,~W.~C., Knowler,~W.~C., \&

Johannes,~R.~S. (1988). Using the ADAP learning algorithm to forecast

the onset of diabetes mellitus. In {\it Proceedings of the Symposium

on Computer Applications and Medical Care} (pp. 261--265). IEEE

Computer Society Press.

The diagnostic, binary-valued variable investigated is whether the

patient shows signs of diabetes according to World Health Organization

criteria (i.e., if the 2 hour postload plasma glucose was at least

200 mg/dl at any survey examination or if found during routine medical

care). The population lives near Phoenix, Arizona, USA.

Results: Their ADAP algorithm makes a real-valued prediction between

0 and 1. This was transformed into a binary decision using a cutoff of

0.448. Using 576 training instances, the sensitivity and specificity

of their algorithm was 76% on the remaining 192 instances.

4. Relevant Information:

Several constraints were placed on the selection of these instances from

a larger database. In particular, all patients here are females at

at least 21 years old of Pima Indian heritage. ADAP is an adaptive learning

routine that generates and executes digital analogs of perceptron-like

devices. It is a unique algorithm; see the paper for details.

5. Number of Instances: 768

6. Missing Attribute Values: Yes

7. Class Distribution: (class value 1 is interpreted as "tested positive for

diabetes")

Class Value Number of instances

0 500

1 268

8. Brief statistical analysis:

Attribute number: Mean: Standard Deviation:

1. 3.8 3.4

2. 120.9 32.0

3. 69.1 19.4

4. 20.5 16.0

5. 79.8 115.2

6. 32.0 7.9

7. 0.5 0.3

8. 33.2 11.8

Due to the relatively low number of records in the training dataset, the accuracy of the system is around 78% and this can be increased with the increase in the training data provided. Excerpts from the training data were at random extracted to be used as testing data.

After all predictions have been made using testing/prediction data, the output is saved to a json file which shows the inputs, expected output and the actual output from the system.

Program Listing

*import* json

*from* numpy *import* loadtxt

*from* keras.layers *import* Dense

*from* keras.models *import* Sequential

*from* keras.models *import* load\_model

*import* numpy *as* np

*# Method to define, evaluate and return a model*

def trainModel():

dataset = loadtxt("training\_dataset.csv", *delimiter*=",")

X = dataset[:,0:8] *# Splits the dataset to get the inputs*

Y = dataset[:,8] *# Splits the dataset to get the output*

model = Sequential()

model.add(Dense(12, *input\_dim*=8, *activation*='relu'))

model.add(Dense(8, *activation*='relu'))

model.add(Dense(1, *activation*='sigmoid'))

model.compile(*loss*='binary\_crossentropy', *optimizer*='adam', *metrics*=['accuracy'])

model.fit(X, Y, *epochs*=150, *batch\_size*=10, *verbose*=0)

*# Test for accuracy of the model*

scores = model.evaluate(X, Y, *verbose*=0)

print("%s: %.2f%%" % (model.metrics\_names[1], scores[1]\*100))

*# model.save("model.h5")*

*return* model

*# Function to write the model to an external*

*# file that can be loaded in the future*

def saveModel(*model*):

*model*.save('model.h5')

*# Function to load model from an external file*

*# and evaluate the accuracy of the model using test data*

def loadModelAndEvaluate():

model = load\_model('model.h5')

model.summary()

dataset = loadtxt("test\_data.csv", *delimiter*=",")

X = dataset[:,0:8] *# Splits the dataset to get the inputs*

Y = dataset[:,8] *# Splits the dataset to get the output*

score = model.evaluate(X, Y, *verbose*=0)

print("%s: %.2f%%" % (model.metrics\_names[1], score[1]\*100))

*# Function to load model from an external file*

*# and make prediction with the model using test data*

def loadModelAndPredict():

model = load\_model('model.h5')

model.summary()

dataset = loadtxt("test\_data.csv", *delimiter*=",")

X = dataset[:,0:8]

Y = dataset[:,8]

*# make class predictions with the model*

predictions = (model.predict(X) > 0.5).astype("int32")

resultsJson = []

*for* ndx *in* range(len(predictions)):

resultsJson.append({

"inputs": X[ndx].tolist(),

"Expected Output": Y[ndx],

"Actual Output": predictions[ndx]

})

resultsJson = json.dumps(resultsJson, *cls*=NumpyEncoder)

*with* open('results.json', 'w') *as* json\_file:

json\_file.write(resultsJson)

*for* ndx *in* range(len(predictions)):

print('Inputs => %s \n Actual Ouput => %d \n Expected Output => %d' % (X[ndx].tolist(), predictions[ndx], Y[ndx]))

*# This encoder will allow us to export our results to a json file*

class NumpyEncoder(json.JSONEncoder):

def default(*self*, *obj*):

*if* isinstance(*obj*, np.integer):

*return* int(*obj*)

*elif* isinstance(*obj*, np.floating):

*return* float(*obj*)

*elif* isinstance(*obj*, np.ndarray):

*return* *obj*.tolist()

*return* json.JSONEncoder.default(*self*, *obj*)

model = trainModel()

saveModel(model)

loadModelAndEvaluate()

loadModelAndPredict()

In conclusion, this system can be used to predict whether patients will get onset diabetes within 5 years or not.