FRENCH-AZERBAIJANI UNIVERSITY

ARTIFICIAL INTELLIGENCE

PROJECT REPORT

Predicting the risk of heart disease in patients

Submitted by:
Emin SULTANOV
Parsha JOARDER

December 18, 2019

Introduction

Aims and Objectives

The purpose of this project is to work on prediction of potential Heart Diseases in patients using Machine Learning approaches(and eventually comparing them) for classifying whether a person is suffering from a heart disease or not, using one of the most used dataset the *"Cleveland Heart Disease"*. Machine learning (ML) has been shown to be effective in assisting in making decisions and predictions from the large quantity of data produced by the healthcare industry. The code is implemented in Python and the two Classification algorithms that was asked to be implemented are the ***Multi-layer perceptron*** and ***Decision Tree***.

The following are the source code files with the corresponding algorithms implemented:

- **mlp.py**
- **decide.py**

The report will present the four concrete tasks required in the lab sheet:

- The "Cleveland Heart Disease" dataset
- Predictions using a Multi-layer perceptron
- Predictions using a Decision Tree
- Comparing MLP and DT

The answers to the questions of each part is clearly presented.Despite that,besides only the answers to the questions at each task there is a clear and vivid explanation how the work has been accomplished and the critical view of the work since it involved an analysis of merits.

1 The "Cleveland Heart Disease" dataset

This part is focused on data pre-processing and exploration,model building will be presented in other parts.The dataset is consisted of 14 attributes(columns) and 303 instances(rows).Luckily no need to handle with null/missing(NaN) values.

1.1 General Questions by studying the dataset:

1. What is the name of the attribute we want to predict ?
   - **target** attribute since it refers to the presence of heart disease in the patient
2. Is that a binary or a multi-class classification ?
   - **Binary classification** since it can take one of just two possible values and the variable to predict "target" is encoded as
     0=absence, 1=presence
3. Which attributes are categorical ?
   - **Categorical attributes:** (whose values correspond to discrete categories and the number of possible values is often limited to a fixed set)

- sex
- chest_pain_type
- fasting_blood_sugar
- rest_ecg ;
- exercise_induced_angina
- st_slope
- num_major_blood_vessels ;
- thalassemia

P.S: "target" has a classification feature type,not categorical

4. How can we encode categorical attributes ?

   - **One-hot encoding** is one of the first encoding schemes taught when we see categorical variables. It maps each level of a category to its own column, and each row has a 1 for the category it belongs to, and a zero otherwise.

1.2 Attribute normalization:

1. Which normalization method will be best adapted so we can preserve the variance of the dataset ?

   - As already mentioned that *gradient descent* is sensitive to numerical attributes,thus one of the effective methods based on standard deviation(variance) is **feature scaling** (known as *min-max normalization*).Moreover gradient descent converges much faster with feature scaling than without it.

2. Will you normalize the data before or after splitting the dataset in training/testing datasets ?

   - Normalization across the instances should be done **after splitting the data** of training and testing set by using only the data from the training set.This is because the test set plays the role of fresh unseen data, so it's not supposed to be accessible at the training stage.Using any information coming from the test set before or during training is a **potential bias** in the evaluation of the performance.

3. Implement a method/function to normalize the attribute using the adequate normalization method.

   The formula of scaling:

$$x\_new = \frac{x - x_{min}}{x_{max} - x_{min}}$$

```python
def normalize(x):
    '''
    Normalization of the data of attributes
    Formula: (x - min)/(max-min)
    x - list to normalize
    '''
    return (x - np.min(x)) / (np.max(x) - np.min(x))
```

Listing 1: Normalization method

2 Predictions using a Multi-layer perceptron

One of the effective methods to predict heart disease is to use the Multi-Layer Perceptron(Artificial Neurons) Training algorithm.Our model involves three different layers.And the accuracy of performance in the network is ranging between 78%-85%.

- **Input Layer:** contains of 13 neurons since thirteen attributes are involved in predicting the heart disease
- **Hidden Layer:** only 5 neurons (also not more than one hidden layer required)
- **Output Layer:** 1 neuron since the prediction is based either on "**0** - no heart disease present" or "**1** - heart disease present".

2.1 Questions

- What are the dimensions of the matrices you will use to represent your model (inputs, parameters and outputs) ? How will you integrate the concept of mini-batch training ?

  - **Input matrix :** $$\underset{242\times13}{I}\left\{\begin{matrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} & x_{1,5} & \cdots & x_{1,13} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} & x_{2,5} & \cdots & x_{2,13} \\ \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots & \cdots\cdots \\ \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots & \cdots\cdots \\ \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots & \cdots\cdots \\ x_{242,1} & x_{242,2} & x_{242,3} & x_{242,4} & x_{242,5} & \cdots & x_{242,13} \end{matrix}\right\}$$

    Since Input matrix is the matrix of xTrain set units.

  - **Output matrix:** $$\underset{242\times1}{O}\left\{\begin{matrix} y_1 \\ y_2 \\ \cdots\cdots \\ \cdots\cdots \\ \cdots\cdots \\ y_{242} \end{matrix}\right\}$$

    Since Output matrix is the matrix of the yTrain set units.

  In case of parameters, we used 2 matrices for weights and 2 matrices for biases. Since the hidden layer has only 5 neurons, the weights for forward propagation from input layer to hidden layer is the matrix 13x5 and the matrix of biases is 242x5. For the case of forward propagation from hidden layer to output layer the matrix of weights is 5x1 since output layer has only 1 neuron and the matrix of biases is 242x1.

  The concept of mini-batch training is used at a time in each epoch.Firstly,the neural network is feeded,the mean gradient of the mini-batch is calculated ,afterwards the weights are updated with the use of mean gradient and finally the gradient is summed over the mini-batch which further reduces the variance of the gradient.

- How should you check whether or not you should keep training your model ?
  - One way to decide when to stop training the model is to monitor the accuracy on the test set; When the validation accuracy steadily decreases but the accuracy on the training set continues increasing, this is a sign of overfitting and therefore the model should not be trained further. The aim is generally to minimize the error thus the algorithm backpropagates across the neural network and updates the weight and it still goes until the error is minimized and stops when the error starts to increase on the test set.

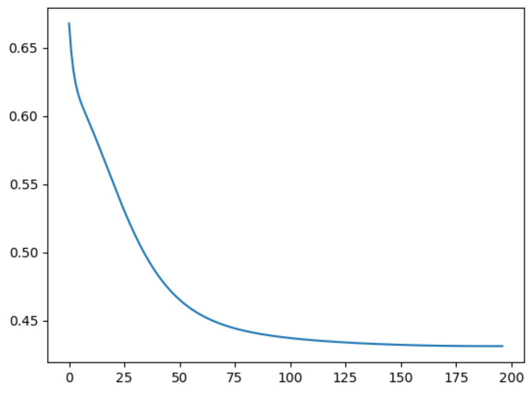

Figure 1: Error decreasing after train epochs

- Draw your network (used the f.f tool: http://alexlenail.me/NN-SVG/)
  - The model is the following:

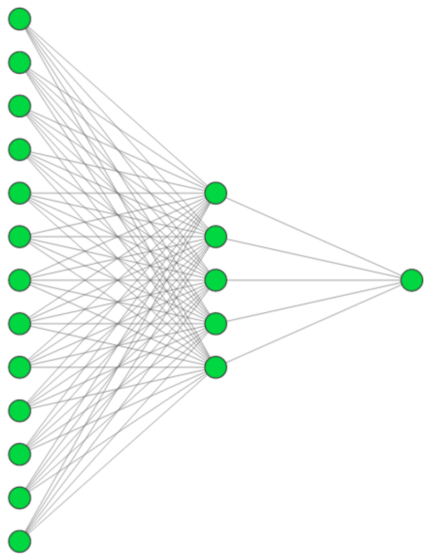

Figure 2: Model of neural network

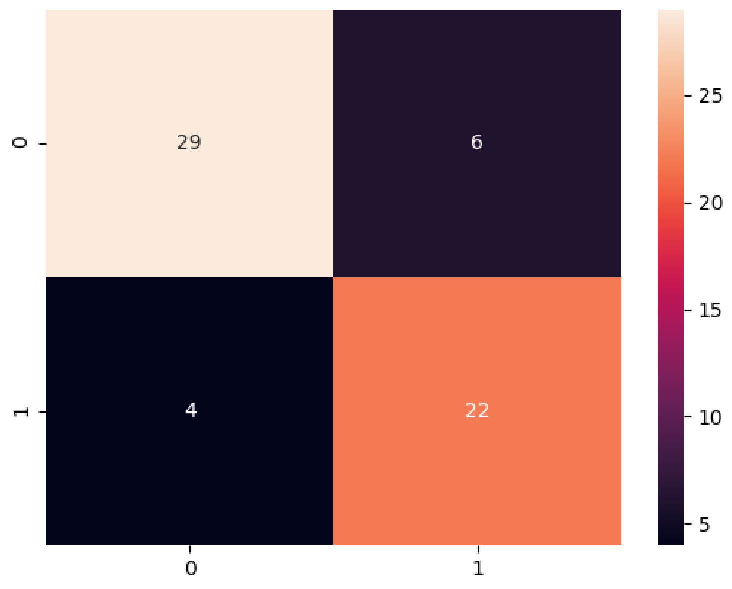

Figure 3: **Confusion Matrix**($\approx$ **84%**$accuracy$)

2.2 Model Evaluation

In order to check the performance of the classifiers(the evaluation metrics) used is **Confusion Matrix**.All the required evaluation has been implemented and computed.

1. the precision of your model
   `def getPrecision()`
   TP/(TP+FP)
2. the accuracy of your model
   `def getAccuracy()`
   (TP+TN)/(FP+TN+TP+FN)
3. the sensitivity of your model
   `def getSensivity()`
   TP/(TP+FN)
4. the specificity of your model
   `def getSpecifity()`
   TN/(TN+FP)

   Definition of Terms:
   • True Positive **(TP)**: subjects who are actually unhealthy and predicted to be unhealthy (29)
   • False Negative **(FN)**: subjects who are actually unhealthy and predicted to be healthy (6)
   • False Positive **(FP)**: subjects who are actually healthy and predicted to be unhealthy (4)
   • True Negative **(TN)**: subjects who are actually healthy and predicted to be healthy (22)

The Confusion Matrix has been constructed above to visualize the prediction results on this classification problem.It displays the correctly predicted as well as incorrectly predicted values by a classifier.

Precision	0.82857
Accuracy	0.83606
Sensitivity	0.87878
Specificity	0.78571

Table 1: **Evaluation of Metrics in MLP**

Question
In the case of predicting the risk of heart disease in patients, would you prefer that your model is sensitive or specific ?

Answer
Well,in our case the model is sensitive since it is more similar to the accuracy precision model,moreover it is better to have sensitive model rather than specific otherwise an individual who does not have a disease should be treated.

2.3 Additional Details about the MLP

- The implementation has been done starting from encoding data with the categorical attributes (`pandas.get_dummies()`).
- Afterwards the data has been split (`def divideData()`) into training and testing sets and normalized (`def normalize()`), as the values of numeric columns in each of sets were in different ranges. The input and target samples are divided into **80%** training set and **20%** testing set.
One training example presented to the 13 network inputs (this is the **feedforward part**).
Weights and bias has been randomly assigned and the learning rate is 0.01.
- As for any gradient descent method (including the mini-batch case), it is important for efficiency of the estimator that each example or minibatch be sampled approximately independently - shuffling the data (`def shuffle_in_unison()`)
- While the error on the test set increases on average during 10 consecutive training epochs then the the training process should be stopped otherwise the errors for all examples in the training set are added together and their sum is back propagated through the network (`def binaryCross()`)
(adjusting the weights of the connections between the processing elements of the network)
- Using a gradient descent learning algorithm the network error should decrease towards a minimum. The processing elements pass the sum of their inputs through a sigmoid function (`def sigmoid()`) *(the activation function)* limiting the neuron output between zero and one.

3 Predictions using a Decision Tree

Applying Decision Tree algorithm to this dataset is a simpler method since each feature is represented by an internal node and further branches are build upon the test conditions and their outcome.
The accuracy using Decision Tree obtained is always 80%

*Note** most of the functions reused in Decision Tree since the initial steps are similar

3.1 Methodology

- Before starting building our decision tree,the model preprocesses the attributes of the dataset by splitting (`def divideData()`) into training and testing sets.Likewise in Multi-Layer Perceptron, **80%** data is taken for training while remaining **20%** data is used for testing.Obviously,we are shuffling our data (`def shuffle_in_unison()`) and normalizing (`def normalize()`) in order to avoid over fitting and have some range of feature values,respectively.

- Since constructing a decision tree is all about finding the attribute that returns the highest discriminative power(information gain) `def discPow()`. Also the information gain uses the entropy measure *(dataset entropy and entropy of specific group)* with `def entropy()`. The attribute with the highest discriminative power is the one that grows the tree in sense of divides the tree into further branches
(`def build()`)

- Using all the mentioned functions, `def predict()` is called where it encodes the testing data with one hot encoding (`def encode()`) and uses `def guess()` in order to verify if it reaches the last node of the tree and if needed goes further.

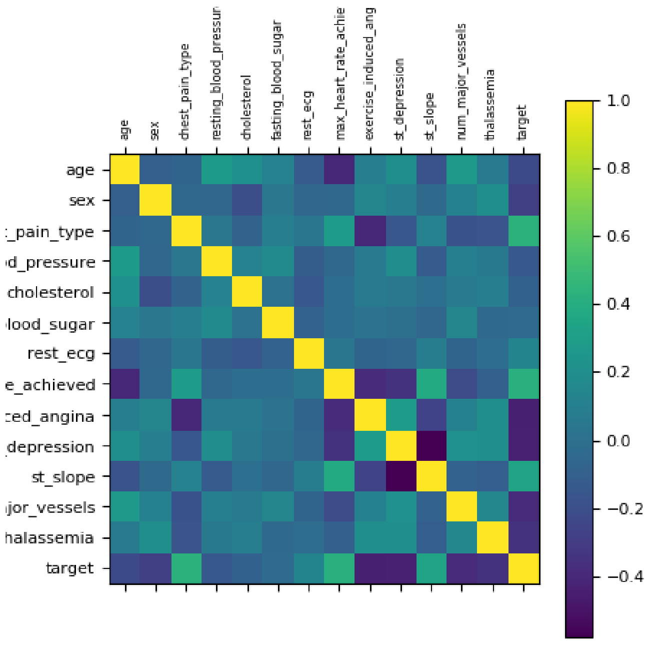

Figure 4: Correlation table

Precision	0.9090
Accuracy	0.8032
Sensitivity	0.7692
Specificity	0.8636

Table 2: **Evaluation of Metrics in DT**

4 Comparing MLP and DT

From the analysis results of the evaluation metrics both in **Table 1** and **Table 2**,
Multi-Layer Perceptron model presented better performance,since it attains the maximum over the output accuracy (83.6%) and proves it is outperformed compared with Decision Tree (80.32%),thus MLP is the best technique to classify in this particular data set.However the prediction accuracy varies according to number of feature sets and number of patient instances.So it is not necessarily MLP being successful every time.

Additionally,we would like to note a case where we retrieved the same value of accuracy(80.32%) both in Multi-Layer Perceptron and Decision Tree,then in that case the sensitivity is studied,since it is a significant factor in medical diagnosis,because it is is the ability of a test to correctly identify those with the disease,otherwise person who has the disease will be identified healthy which may lead to serious consequences.

Finally,in our opinion the efficiency of algorithms not only depends on output accuracy, but it also depends on other factors such as the time taken to execute,amount of memory resources used to execute etc.