Diabetes Diagnosis by Machine Learning using acquired data set

CSE 684 - Term Project Final Report

Instructor: Dr. Andrew Lee

Burak Kakillioglu

Rohit Sharma

Varun Jindal

December 15th, 2015

Contents

The purpose of this project is to do a comparative analysis of different machine learning techniques by diagnosing Diabetes over an acquired Pima Indians Data Set using each one of them. This study will tell us how these machine learning methods outperform each other and which one is better to do this type of diagnosis. We have compared Neural Network, Classification Naïve Bayes Classifier, and KNN (K-Nearest Neighbor Classifier) methods to diagnose Diabetes over Pima Indians Data Set. There are 768 records in our dataset, out of which 500 are healthy patients and 268 are diabetic. We have divided the data set into training data and test data. The percentage of division varies as per the method used. Using Neural Network we were able to achieve highest accuracy of 95.4%. Using Naïve Bayes Classifier we could achieve 78.8% accuracy. KNN Classifier gave us an accuracy of 73.3%.

Motivations, background, and introduction:

There are various applications where machine learning can be applied to reduce the work required by humans and equivalent or even better results can be produced. Diagnosis of Diabetes in humans is one such application where it's possible to reduce the human effort to the basic diagnosis and differentiate between diabetic and healthy patients. This was the main motivation for us to pursue this project.

Doctors test patients to look for some particular symptoms while diagnosing Diabetes. Same symptoms can be used as parameters in different machine learning techniques and the system can be trained to identify if the patient has Diabetes or is healthy based on those symptoms.

This is what we are trying to do here. We acquired a Diabetes Data Set for females of at least 21 years old of Pima Indian heritage. We applied three machine learning techniques, namely Neural Network, Naïve Bayes Classifier, and KNN Classifier over this acquired data set and did a comparative analysis of all these method and how they are performing to identify Diabetes in new dataset.

Problem Statement:

To do the Diabetes diagnosis over an acquired data set using different machine learning methods, analyze their results, and make an informed comparison of those methods using the results.

Core Contents:

Neural Network:

Neural network is one of the most popular machine learning techniques. As it carries the word "Neural" on its name, the neural networks are inspired by the structure of biological neural systems although the operations of neural networks and nervous systems are not similar. A neural network is built up highly interconnected neurons working together as in the brain to solve specific problems that may be too complex to be understood by human beings or other computational techniques.

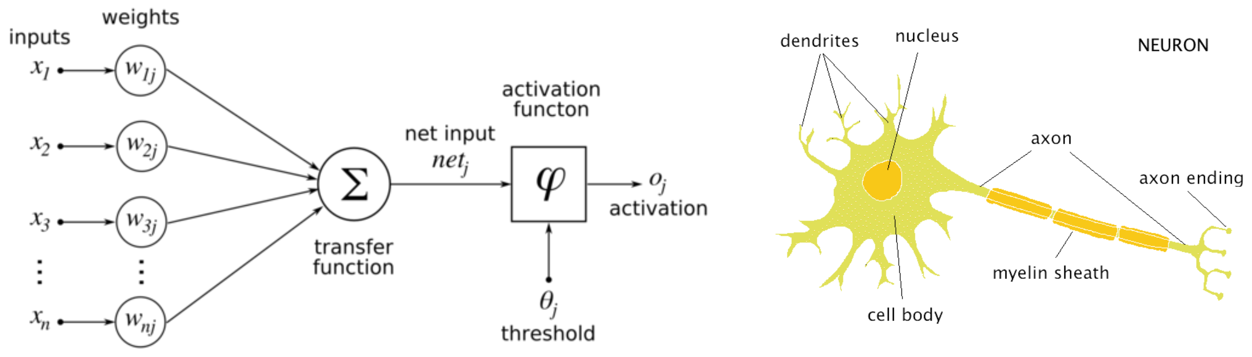


Figure 1: Neuron structure in neural networks (left) and in nervous systems (right)

Neural networks operate different than traditional computational methods. Traditional methods solves problems by following certain cognitive procedure which are precisely defined by the programmer. Those methods, therefore, perform a specific task and are predictable. However, a neural network has its own way of solving a problem. It does not perform task to solve the problem like traditional methods. It learns how to solve the problem using a *deep learning* technique by observing previous examples. Although the results of neural networks are generally unpredictable, they are capable of deriving meanings from complex problems.

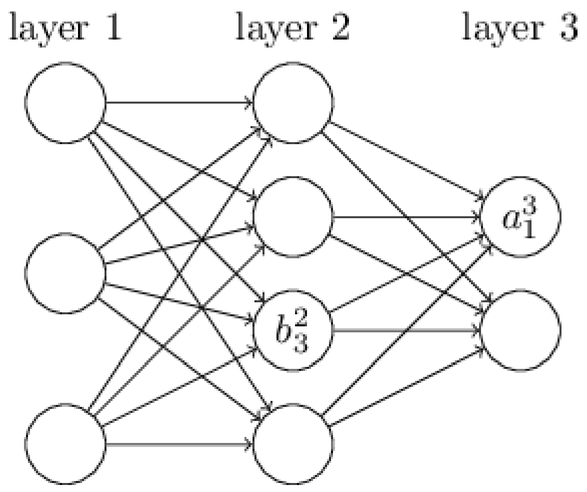


Figure 2: A simple neural network structure [2]

A neural network consists of layers and layers consist of neurons. A neural network can be defined basically by the weights of each neuron connection, the connection structure of the neurons between layers, and activation function. Figure 2 shows a simple network structure with 3 layers. Activations of each neuron is determined by all neurons in previous layer and weights between those neurons. Since the first layer is input layer that we know the activations of input neurons, we can calculate the activation of all neurons including output layer. This procedure is called *forward propagation* which is the actual operation of a neural network. The activation of the 3rd neuron in layer 2 depicted in Figure 2 is calculated with following formula in forward propagation:

$$a_j^l = \sigma\left(\sum_k \left[w_{jk}^l a_k^{l-1} + b_j^l\right]\right) \quad [2]$$

a_j^l: $Activation\ of\ j^{th}\ neuron\ in\ layer\ l$

b_j^l: $Bias\ of\ j^{th}\ neuron\ in\ layer\ l$

w^l: $The\ weights\ matrix\ that\ connects\ (l-1)^{th}\ layer\ to\ l^{th}\ layer$

σ: $The\ actication\ funciton$

Work done:

In this project we compared different machine learning techniques on *diabetes diagnosing* task. We used Pima Indians Diabetes Database of UCI Machine Learning Repository as our samples. We have done classification using 8 attributes which are:

1. Number of times pregnant
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. Diastolic blood pressure (mm Hg)
4. Triceps skin fold thickness (mm)
5. 2-Hour serum insulin (mu U/ml)
6. Body mass index (weight in kg/(height in m)^2)
7. Diabetes pedigree function
8. Age (years)

Our neural network classifies samples as "Diabetic" or "Healthy" based on attributes above. So in this case, we have 8 attributes and 2 classes which means our neural network has 8 neurons in the input layer (input neurons) and 2 neurons in the output layer (output neurons). In our project we used 1 hidden layer other than input and output layers with size 25.

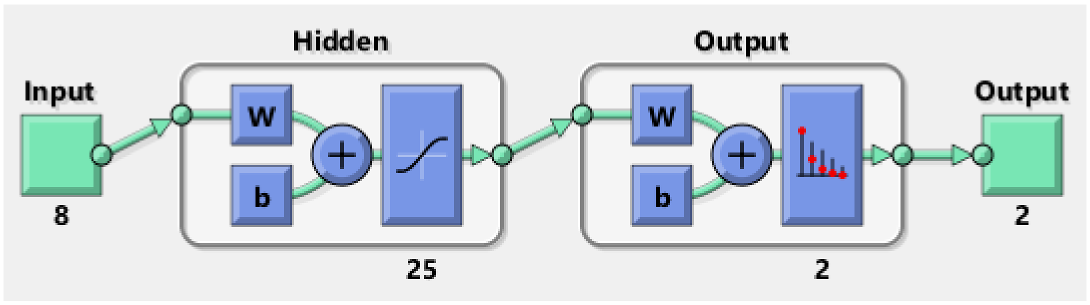


Figure 3: Neural network for diabetes diagnosis

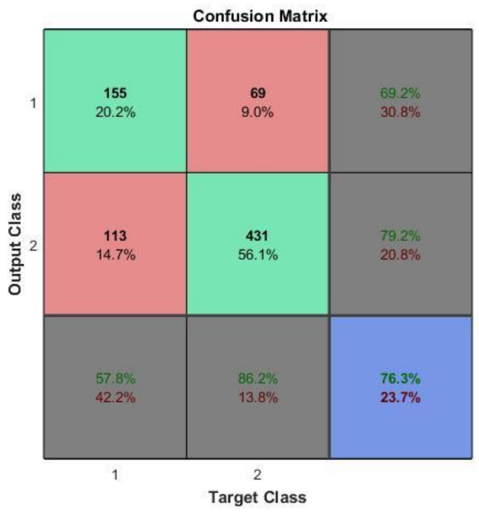
Experiments:

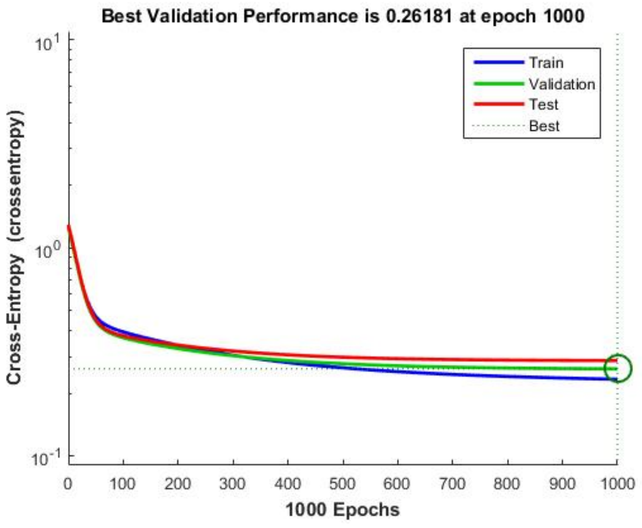
We used MATLAB Machine Learning Toolbox as our experiment platform since it provides lots of built-in features including various training algorithms for neural networks.

We acquired the dataset and divide it into 3 pieces: 70% for training, 15% for testing, and 15% for validation. Then we created the network structure with input layer, output layer, and 1 hidden layer that has 25 neurons as depicted in Figure 3.

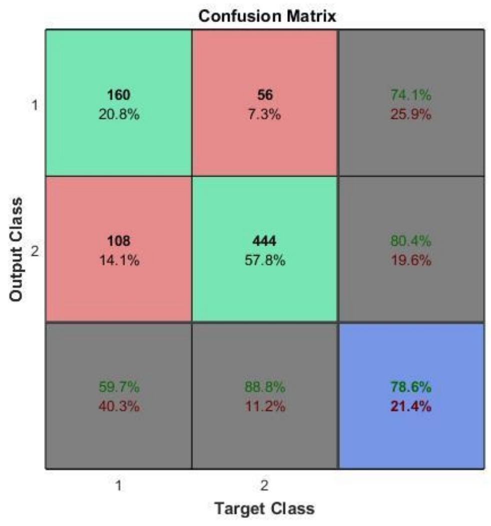
After creating the network and obtaining the data, we trained our model using various deep learning techniques. Those techniques include Levenberg-Marquardt backpropagation, Bayesian Regularization, BFGS Quasi-Newton backpropagation, Gradient Descent, and Variable Learning Rate Gradient Descent.

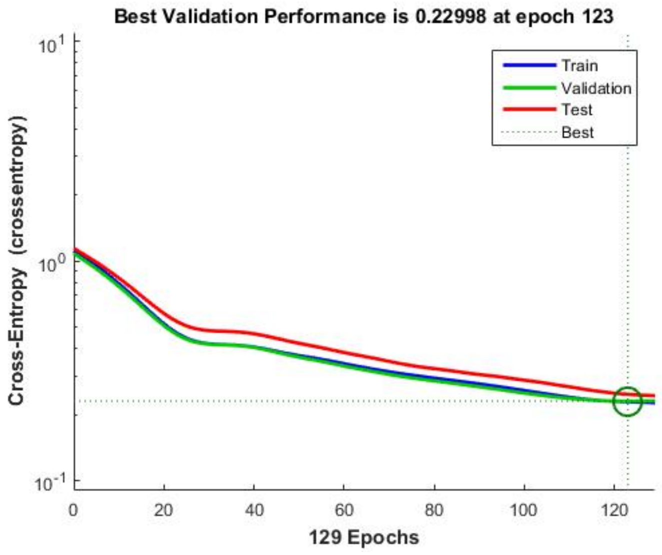
Gradient Descent:



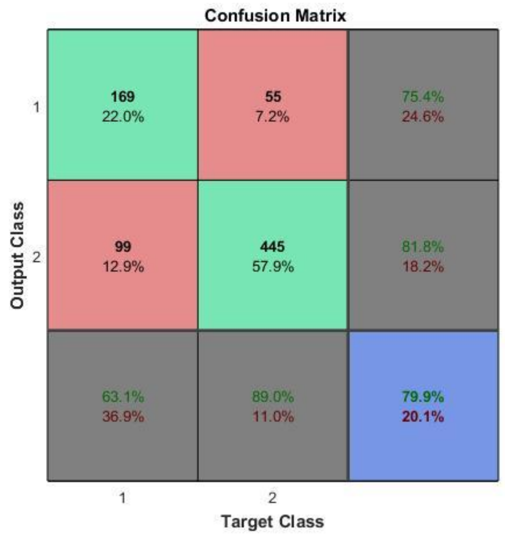


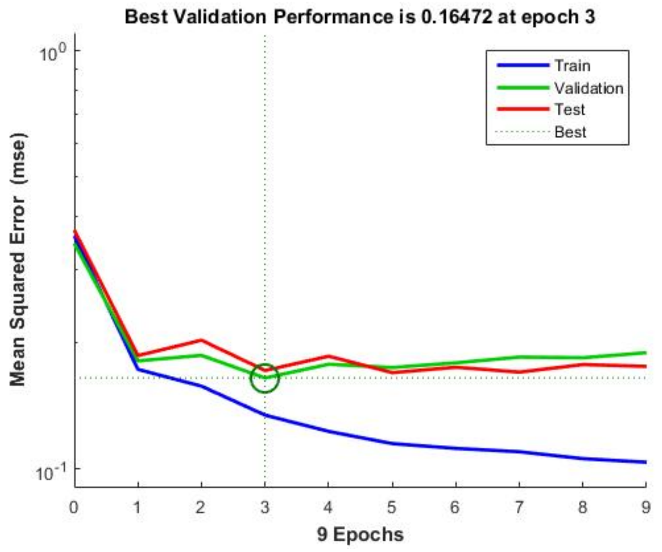
Gradient descent with variable learning rate:



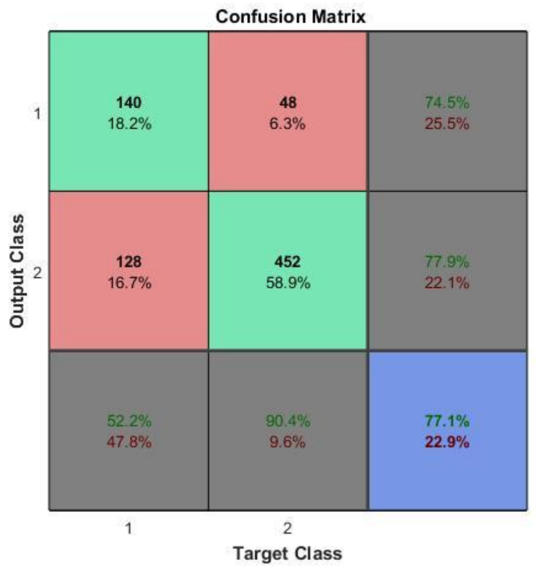


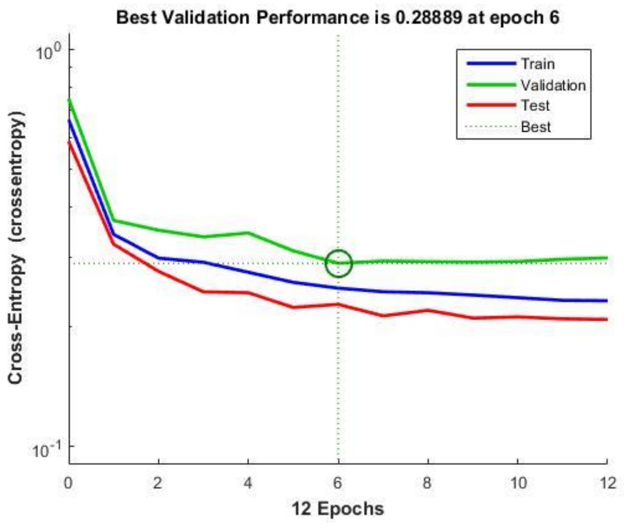
Levenberg-Marquardt backpropagation:





BFGS Quasi-Newton backpropagation:

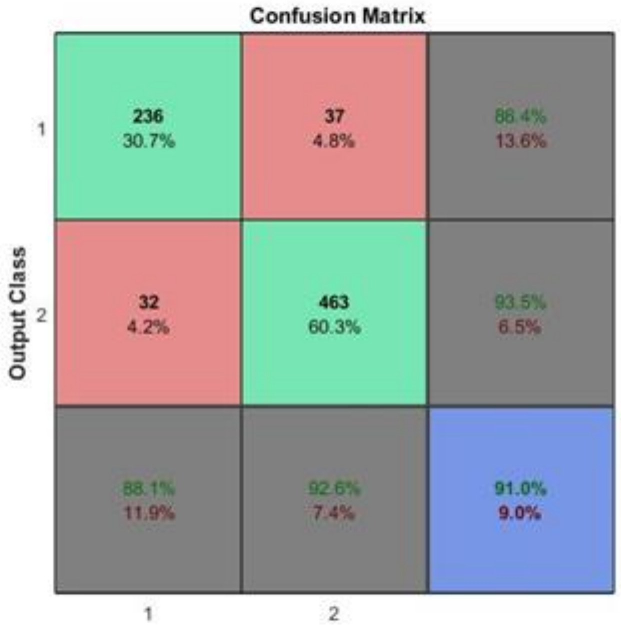


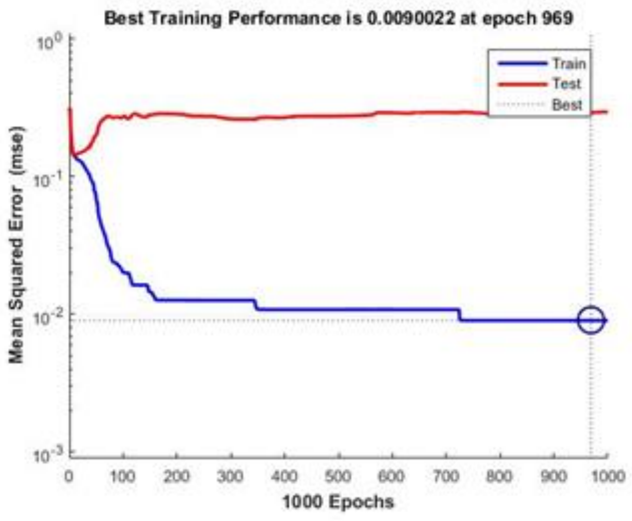


Bayesian regularization:

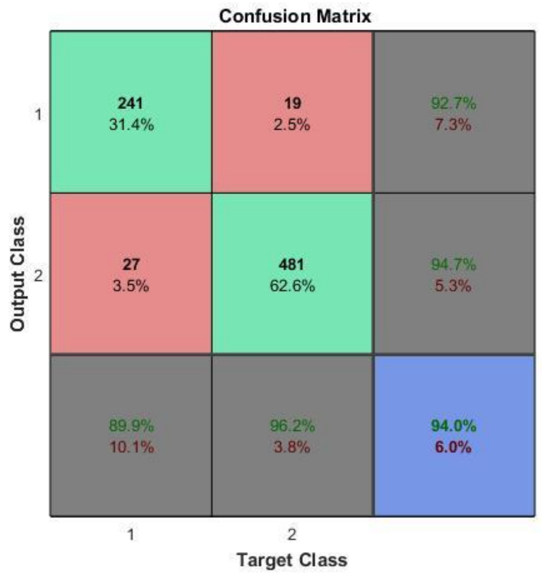
Since this technique does not use validation data, we divide our dataset into 2 parts instead of 3.

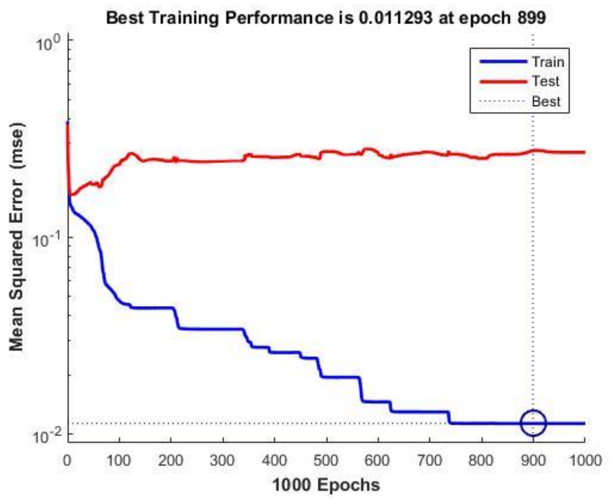
70% for training and 30% for testing:



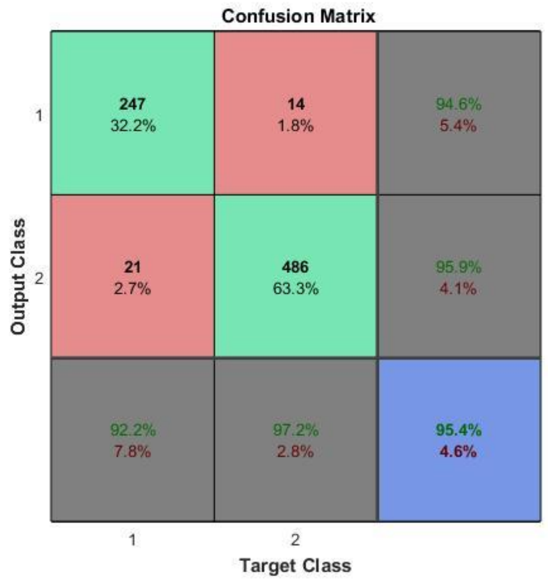


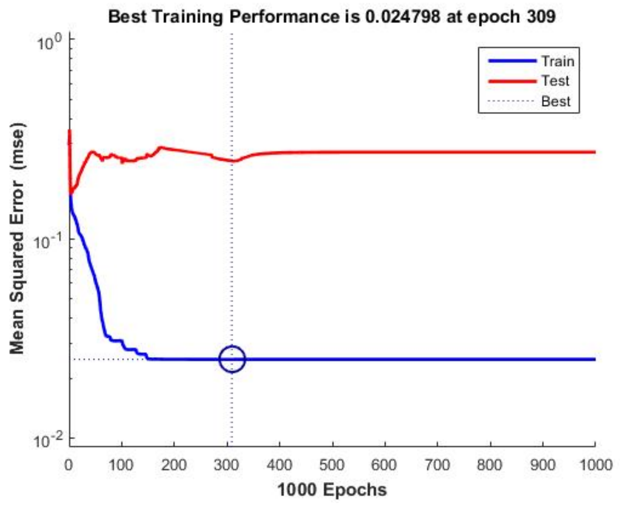
80% for training and 20% for testing:





90% for training and 10% for testing:





Classification Naïve Bayes Classifier:

There are various methods of performing machine learning over acquired data sets to achieve desired accuracy level in training the model. One of those, widely known, methods is Naïve Bayes Classification. In this project report we are comparing the results achieved by various machine learning techniques and our conclusion of those results over an acquired Diabetes Data Set. In this section we are going to address Naïve Bayes Classification for the same.

What are Naïve Bayes Classifiers?

Naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features. In this project we have used Classification Naïve Bayes classifier supported by MATLAB.

Naive Bayes is a classification algorithm that applies density estimation to the data.

The algorithm leverages Bayes theorem, and (naively) assumes that the predictors are conditionally independent, given the class. Though the assumption is usually violated in practice, naive Bayes classifiers tend to yield posterior distributions that are robust to biased class density estimates, particularly where the posterior is 0.5 (the decision boundary).

Naive Bayes classifiers assign observations to the most probable class (in other words, the maximum a posteriori decision rule). Explicitly, the algorithm:

1. Estimates the densities of the predictors within each class.
2. Models posterior probabilities according to Bayes rule.
3. Classifies an observation by estimating the posterior probability for each class, and then assigns the observation to the class yielding the maximum posterior probability.

In MATLAB Classification Naïve Bayes classifier is exposed using function **fitcnb.** We can train a Classification Naïve Bayes classifier by providing training data to **fitcnb** function. This fitcnb function returns a multiclass Naïve Bayes Model trained by the predictors provided in the training data. This multiclass Naïve Bayes Model can further be used to predict results on the test data.

In this project, we have used Pima Indian Diabetes Data Set for testing our machine learning techniques and to derive conclusions about their efficiency. As part of the final source code, I have taken first 60% records in the data set as training data and next 30% records as testing data.

I have trained the Classification Naïve Bayes classifier model using this training data, providing training labels as D and H where D and H are our class i.e. possible outputs. Here D represent 1 in the output results i.e. test positive for diabetes and H represent 0 in the output results i.e. tested negative for diabetes.

While training the data it is important to provide the predictors and prior probability as well (if known in advance). Predictors are the features or in colloquial language input parameters in the dataset that are being used by CNB classifier to generate a prediction model.

In my general research about Classification Naïve Bayes classifiers I found that they are usually very accurate and require less amount of training occurrence to get an optimally trained model. Hence, to verify these claims I tweaked with various features associated to these classifiers.

Observation 1:
Changing the number of training and test records changes the accuracy of the test results significantly for Classification Naïve Bayes classifier.

Test Results while keeping first 70% records as Training Data and next 30% records as Testing Data:

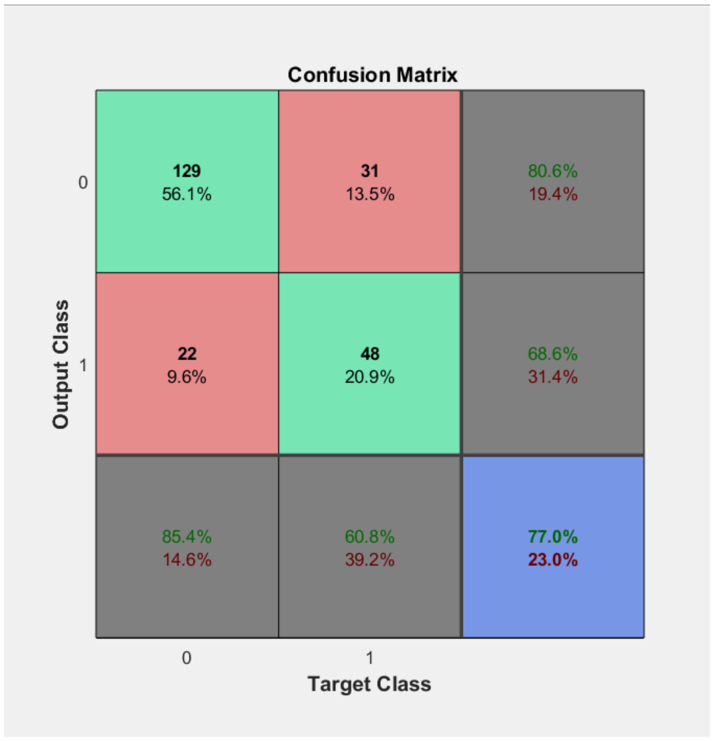


Figure 4- Confusion matrix for 70% training records

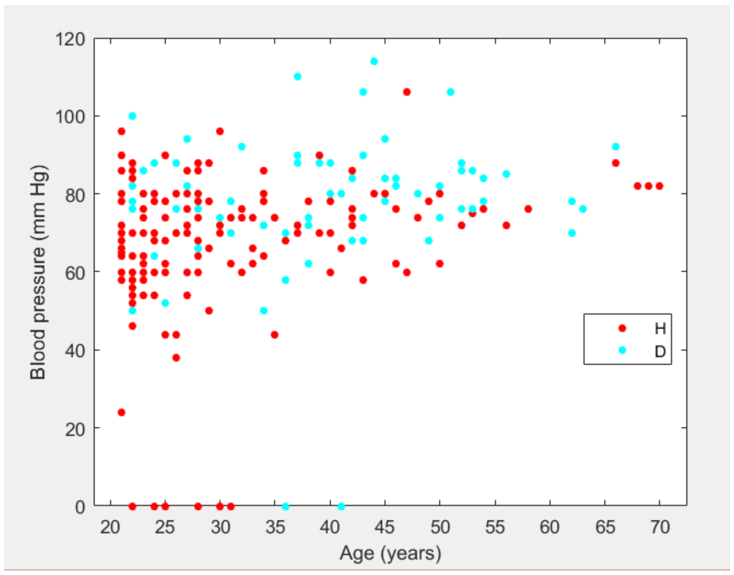
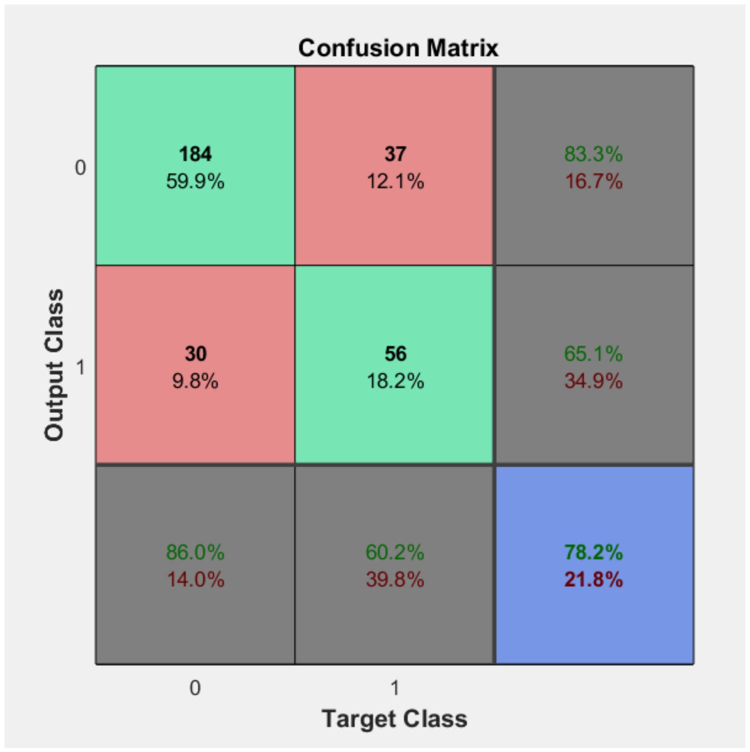


Figure 5- Scatter Plot for 70% training records

Test Results while keeping first 60% records as Training Data and next 40% records as Testing Data:


Confusion Matrix
Output Class
Target Class

	0	1	
0	184 59.9%	37 12.1%	83.3% 16.7%
1	30 9.8%	56 18.2%	65.1% 34.9%
	86.0% 14.0%	60.2% 39.8%	78.2% 21.8%

Figure 6- Confusion matrix for 60% training records

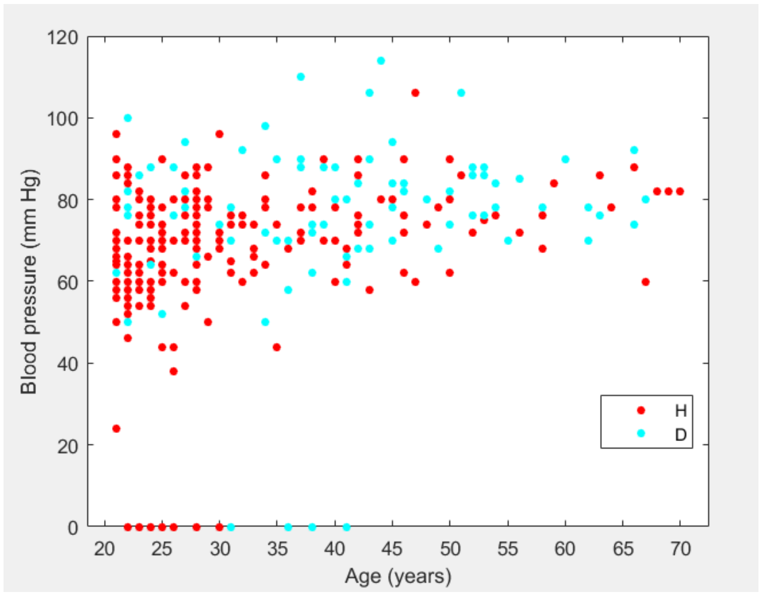


Figure 7- Scatter Plot for 60% training records

As it's evident from the above test results that, accuracy increased from 77% to 78.2% for decrease in Training Records from 70% to 60% i.e. increase in Testing Record from 30% to 40%. However, further decrease in Training Records only worsened the accuracy of the Test Results. These were the optimal values that could be found for the Pima Indians Test Data by changing the training records.

Observation 2:

MATLAB documentation suggests that in cases where we are fully aware of the data set and have full access to it, providing already known probabilities of getting classes (test results) will have a large positive impact on the predictions provided by the Naïve Bayes Model. In my testing, though I did not find very significant impact of providing the known test results probabilities to the classification model while training, still I did find some performance improvements.

In our training data, 38% people were diabetic and 62% people were healthy. I set the prior attribute of the Classification Naïve Bayes Model as the probability of being diabetic to 0.4 and probability of being healthy to 0.6 and got the difference in the accuracy of the predictions. I also tested with other near ranges of probabilities like [0.5 0.5] but after that performance only decreased.

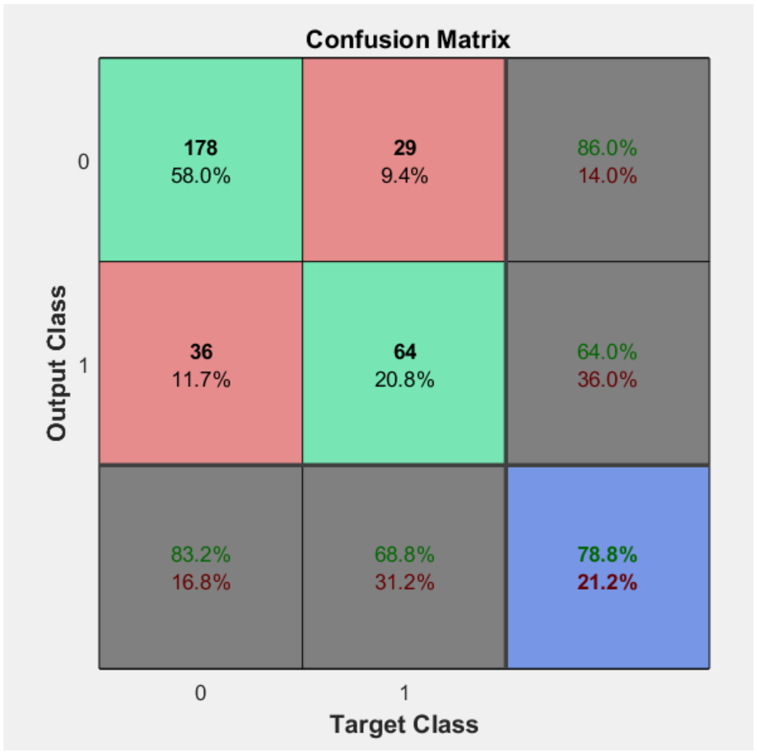


Figure 8-Confusion Plot for Prior Prob. 0.4 and 0.6

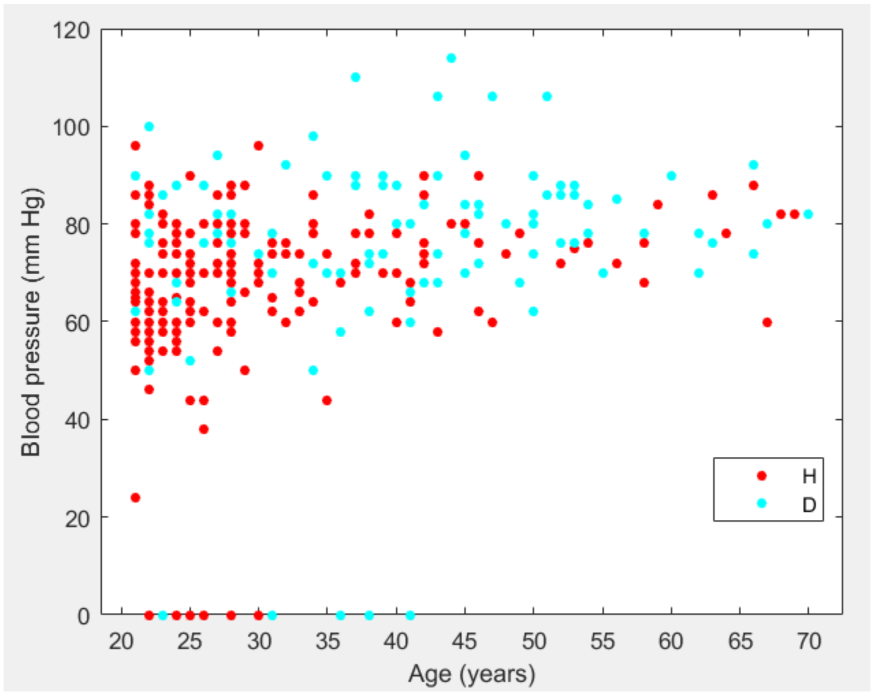


Figure 9 - Scatter graph for second change - prior probabilities

Final best accuracy achieved was 78.8% as per the confusion matrix with 60% test data and 0.4 and 0.6 prior probabilities.

KNN Classifier:

Introduction

- Supervised Learning classifier.
- KNN is a non-parametric lazy learning algorithm: **Non parametric** - means that it does not make any assumptions on the underlying data distribution. This is pretty useful, as in the real world, most of the practical data does not obey the typical theoretical assumptions made (e.g. Gaussian mixtures, linearly separable etc.)
  Lazy algorithm: It does not use the training data points to do any generalization. In other words, there is no explicit training phase or it is very minimal. This means the training phase is pretty fast. Lack of generalization means that KNN keeps all the training data. More exactly, all the training data is needed during the testing phase.

Assumptions:

KNN assumes that the data is in a feature space. More exactly, the data points are in a metric space. The data can be scalars or possibly even multidimensional vectors. Since the points are in feature space, they have a notion of distance.

Types of **Distances**:

'euclidean' — Euclidean distance (default)

'cityblock' — Sum of absolute differences

'cosine' — One minus the cosine of the included angle between points (treated as vectors)

'correlation' — One minus the sample correlation between points (treated as sequences of values)

'hamming' — Percentage of bits that differ (suitable only for binary data)

We are also given a single number "k”. This number decides how many neighbors (where neighbors are defined based on the distance metric) influence the classification. This is usually an odd number if the number of classes is 2. If k=1, then the algorithm is simply called the nearest neighbor algorithm.

Parameters:

Sample: Matrix whose rows will be classified into groups. Sample must have the same number of columns as Training

Training: Matrix used to group the rows in the matrix Sample. Training must have the same number of columns as Sample. Each row of Training belongs to the group whose value is the corresponding entry of Group.

Group: Vector whose distinct values define the grouping of the rows in Training.

K: The number of nearest neighbors used in the classification.

Distance:

- 'euclidean' — Euclidean distance (default)
- 'cityblock' — Sum of absolute differences
- 'cosine' — One minus the cosine of the included angle between points (treated as vectors)
- 'correlation' — One minus the sample correlation between points (treated as sequences of values)
- 'hamming' — Percentage of bits that differ (suitable only for binary data)

Rule:

- 'nearest' — Majority rule with nearest point tie-break (default)
- 'random' — Majority rule with random point tie-break
- 'consensus' — Consensus rule

Advantages of using K nearest neighbor method:

- KNN's decision boundary can take on any form
  This is because KNN is non-parametric, i.e. it makes no assumption about the data distribution. Contrast this to NB, which assumes that attributes are conditionally independent to each other given the class, and are normally distributed
- KNN can handle zero frequency problem
  You have a zero frequency problem when you don't observe a count-valued attribute for a particular class. This is a common problem in text classification, for example in Naïve Bias classifiers any email which contains "password" will always be classified as spam, the reason being P(password|non-spam) = 0, but in real a non-spam email can also contain the phrase “password”. KNN will just search for result on basis on nearby data and will be able to classify it as non-spam.

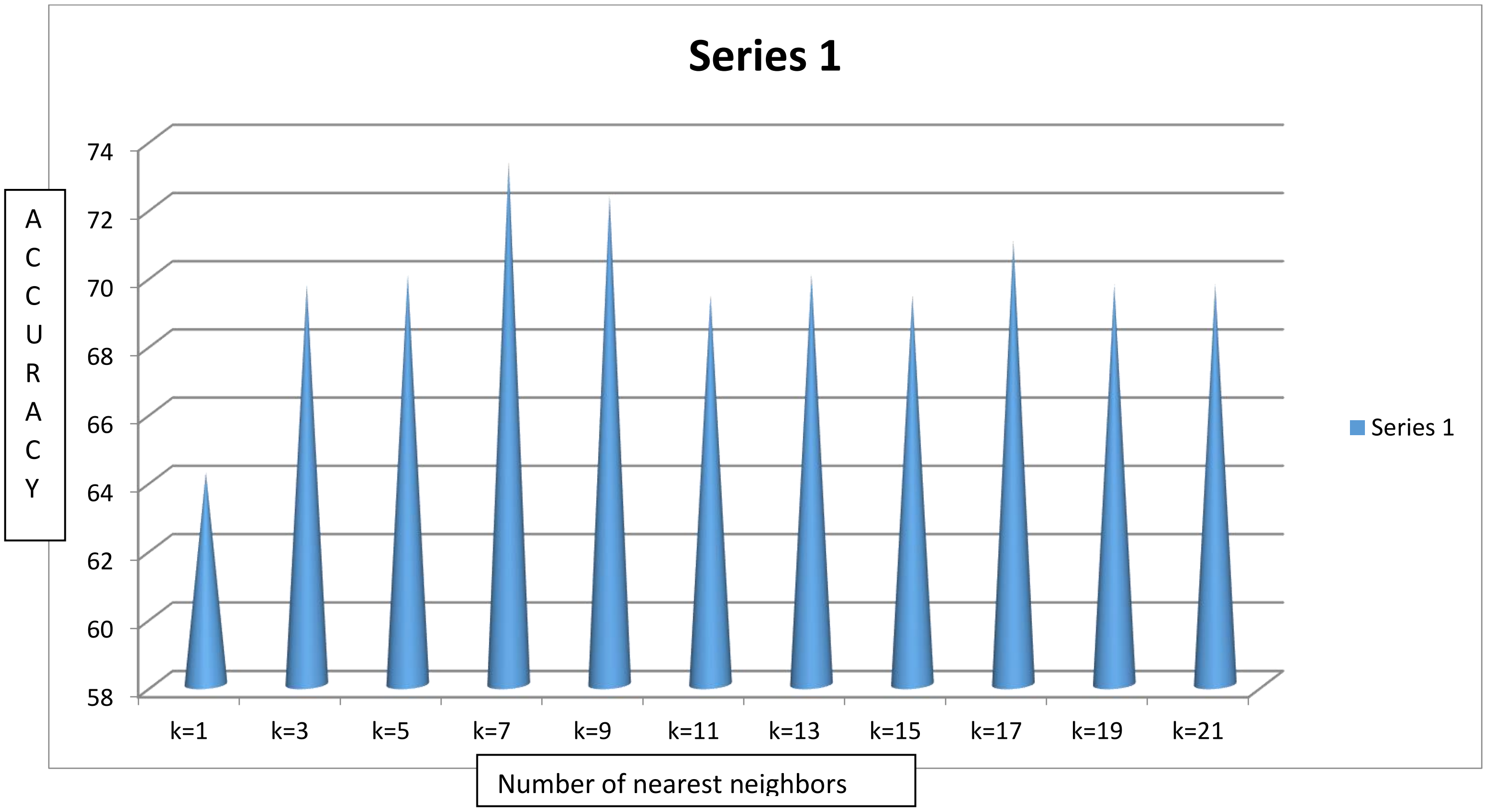
Limitations of KNN

- KNN doesn't know allow weighted attributes i.e. which attributes are more important when computing distance between data points (usually Euclidean distance), each attribute normally weighs the same to the total distance. This means that attributes which are not so important will have the same influence on the distance compared to more important attributes.
- Doesn't handle missing data gracefully
  With KNN, we can't do classification if we have missing data. The reason is that distance is undefined if one or more of attributes (which are essentially dimensions) are missing, unless we are omitting these attributes manually when computing distance. We thus have to rely on common solutions for missing data, e.g. imputing average values.

Trend:

Keeping : Distance : Euclidean & Rule : Nearest

Variable : K = {1 to 21}



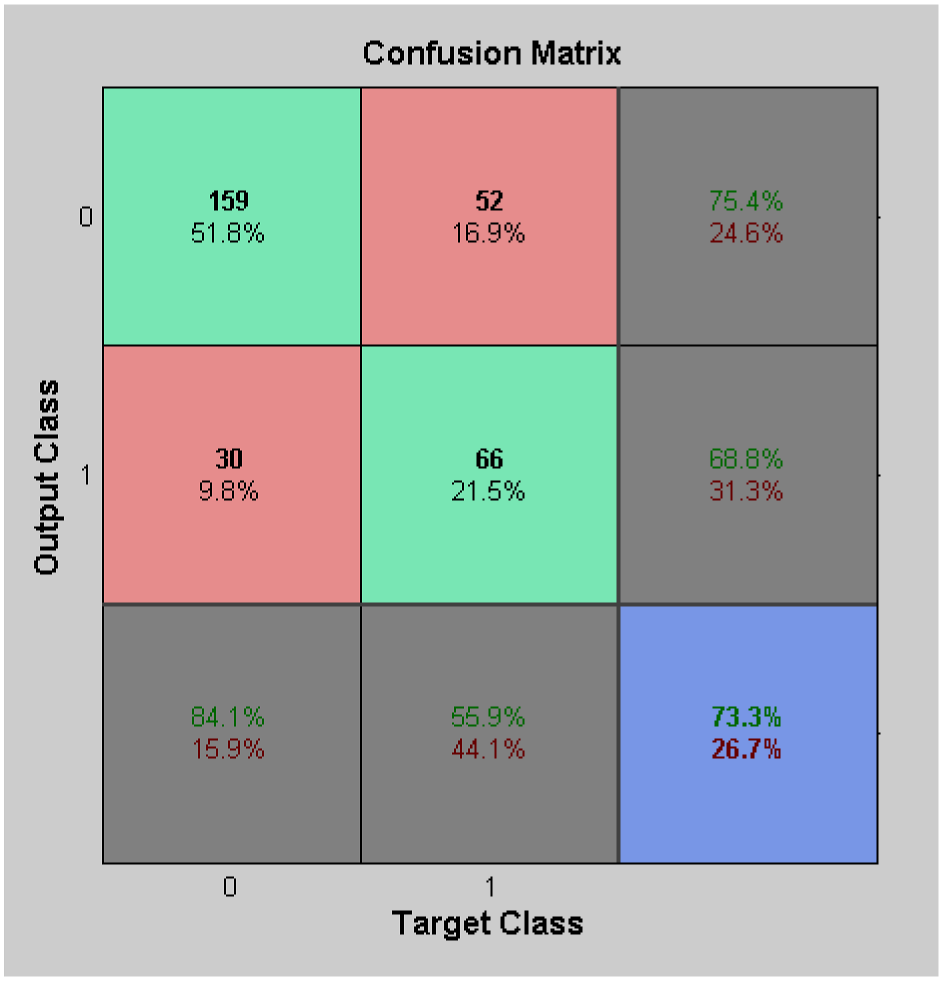
Confusion matrix when:

K = 7

Distance : Euclidean

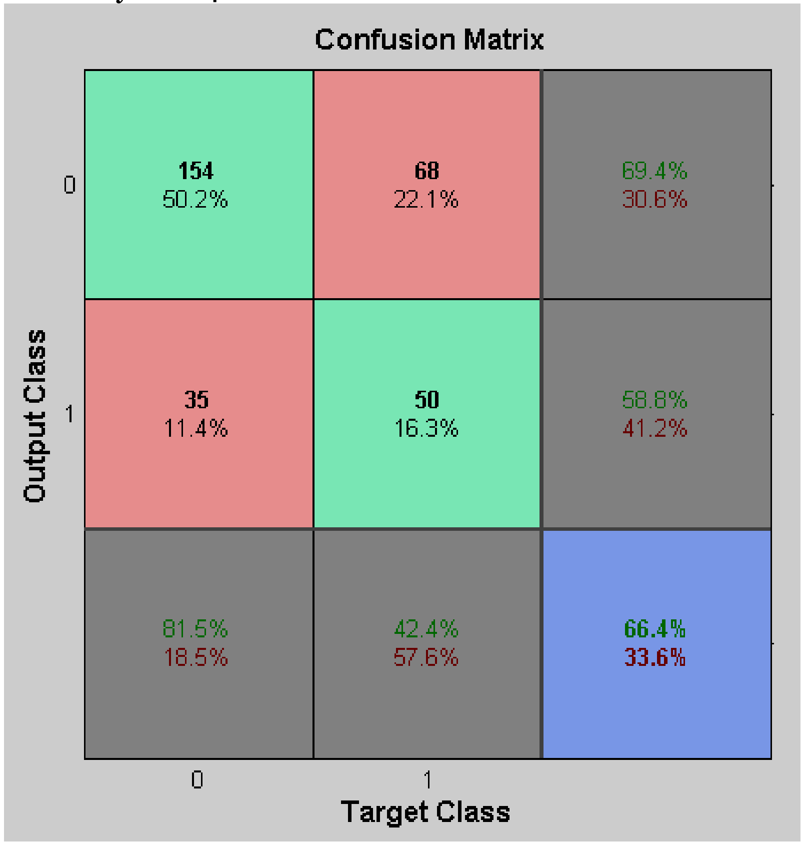
Rule : Random

Accuracy : 73.3% ---- same as when Rule was 'Nearest'

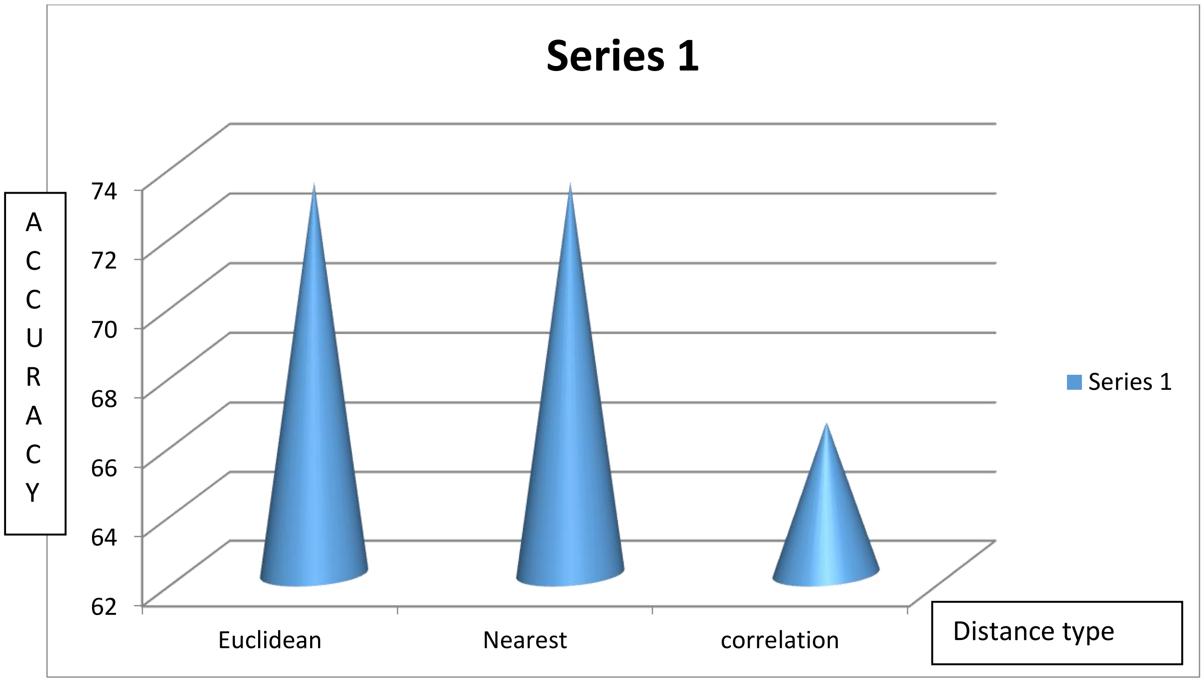


Confusion matrix when:

K = 7
Distance : correlation
Rule : Random
Accuracy : 66.4%



K = 7, Rule : Random



Conclusions and Future Work:

We have compared three methods for analyzing Pima Indian Dataset, Neural Network, Classification Naïve Bayes classifier, and KNN classifier. After analyzing all these results we came to the conclusion that Neural Network's Bayesian regularization with 90-10 ratio is the best method with an accuracy of 95.4%.

Among these methods separately, Bayesian regularization is best for neural network, CNB classifier with prior probabilities achieve best results in CNB, and KNN with k=7 and city block distance achieved best results for KNN classifier.

Neural Network:

Method	**Accuracy %**	**Time (sec)**
Gradient Descent	76.3	3.51
Gradient Descent with variable learning rate	78.6	1.39
Levenberg-Marquardt backpropagation	79.9	1.13
BFGS Quasi-Newton backpropagation	77.1	1.71
Bayesian regularization (70-30)	91.0	32.31
Bayesian regularization (80-20)	94.0	23.14
Bayesian regularization (90-10)	95.4	28.45

In this work, we created a neural network model to diagnose diabetes. We used UCI Pima Indians Dataset to train and test our model. Our neural network model has a 3 layer structure: Input layer, output layer, and 1 hidden layer with 25 neurons. As deep learning procedure, we compared different training algorithms including; gradient descent, gradient descent with variable learning rate, Levenberg-Marquardt backpropagation, BFGS Quasi-Newton backpropagation, and Bayesian regularization. By using the first techniques we get 76.3% to 79.9% accuracy which is an acceptable level. However when we try Bayesian regularization, the accuracy increased to 91% which was above our expectations. Then we try to further increase that amount by increasing the training data amount among the whole dataset we have. As we increase the ratio that number of sample in training set over number of sample in testing set the accuracy increased up to 95.4%. We concluded that the neural network that is trained using Bayesian regularization is giving the best accuracy. However it is also the most costly learning with respect to time-efficiency.

Classification Naïve Bayes Classifier:

In this work, we trained Classification Naïve Bayes classifier to diagnose diabetes over Pima Indians Dataset. We tested with below three configurations. First, we trained CNB classifier with 70% data and were able to achieve 77% accuracy after testing it with remaining 30% data. It took 1.17 second to train the classifier. Then we try to further increase the accuracy by increasing the testing data to 40% and trained the network with 60% of dataset. We were able to achieve 78.2% of accuracy with data. It took 1.19 second to train the network in this case. If we know the prior probabilities of the classes in the training data then we can increase the accuracy of the CNB classifiers. In our dataset there are 35% data which are diabetic and 65% which are healthy. We provided 0.4 prior probability of being diabetic and 0.6 of being healthy. We were able to achieve 78.8% accuracy with training time as 2.15 which is respectable time for achieving this level of accuracy.

Method	**True Positive Ratio %**	**Time (Sec)**
CNB classifier (70-30)	77.0	1.170327
CNB classifier (60-40)	78.2	1.193868
CNB classifier with prior probability (0.4-0.6)	78.8	2.159065

Prior Prob	Loss
Loss in model with prior prob (0.4-0.6)	0.2752
Loss in model with prior prob (0.5-0.5)	0.30302

There are various other methods to increase the performance of Classification Naïve Bayes (CNB) classifier. Few of them which are known to be quite effective and are popular like 'The Fisher Method', Decision Tree Hybrid, and **crossval** Cross-Validation Naïve Bayes Classifier methods can be used to test the effects on accuracy of the CNB classifier over Pima Indians Diabetes Test Data.

KNN Classifier:

In order to compare the various combinations possible to evaluate the performance of the KNN classifier method, we have compared the accuracy of the algorithm for different value of neighbors.

Comparison for value k=1 to k=21, it can be seen that highest accuracy is achieved for K=7. Keeping K=7 further computations are performed.

From the perspective of the time taken to classify the data, K=7 and distance ='city block' has the best results, this is because of the fact that the data set has less scattered space.

Method	**True Positive Ratio %**	**Elapsed Time**
K=7 | Euclidean | Random	73.3	0.471 seconds
K=7 | Correlation | Random	66.4	0.785 seconds
K=7 | City Block | Random	72.3	0.255 seconds

References:

Neural Network:

1. https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html#Introduction to neural networks
2. http://neuralnetworksanddeeplearning.com/
3. https://en.wikipedia.org/wiki/Artificial_neural_network

Classification Naïve Bayes Classifier:

4. Naïve Bayes Classifier - Wiki:

https://en.wikipedia.org/wiki/Naive_Bayes_classifier

5. Matlab documents – Classification Naïve Bayes

http://www.mathworks.com/help/stats/classificationnaivebayes-class.html

6. Matlab documents – Classification Naïve Bayes – Cross Validation

http://www.mathworks.com/help/stats/classificationnaivebayes.crossval.html

7. Ways to improve accuracy of Naïve Bayes classifier.

http://stackoverflow.com/questions/3473612/ways-to-improve-the-accuracy-of-a-naive-bayes-classifier

8. Decision-Tree Hybrid

http://robotics.stanford.edu/~ronnyk/nbtree.pdf

9. Pima Indians Diabetes Data Set - https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes

KNN Classifier:

10. Matlab documents : http://www.mathworks.com/help/bioinfo/ref/knnclassify.html
11. Naïve Bayes Classifier - Wiki: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
12. Ways to improve accuracy of KNN classifier : http://www.fon.hum.uva.nl/praat/manual/kNN_classifiers_1_1__Improving_classification_accuracy.html
13. Pima Indians Diabetes Data Set - https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes

Appendix:

Neural Network:

```
clear all; clc

%% Prepare data
pima = csvread('pima-indians-diabetes.data');
pimaInputs = pima(:,1:8).';
pimaOutputs = [pima(:,9) 1-pima(:,9)].';

inputs = pimaInputs;
targets = pimaOutputs;

%% Create a Pattern Recognition Network
hiddenLayerSize = [25];
net = patternnet(hiddenLayerSize);

%% Set up Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

%% Train the Network
tic
[net,tr] = trainbr(net,inputs,targets);
toc

%% Test the Network
outputs = net(inputs);
errors = gsubtract(targets,outputs);
performance = perform(net,targets,outputs)

%% Plots
figure, plotperform(tr)
figure, plotconfusion(targets,outputs)
```

Naïve Bayes Clasifier:

```
clear all; clc
%% Load Data
pimaData = csvread('pima-indians-diabetes.data');

%% TrainCount
trainCount = round(length(pimaData)*0.6); % 60% of data is for training

%% Train
trainData = pimaData(1:trainCount,1:8);
trainLabels = char('D'*pimaData(1:trainCount,9) + 'H'*(1-pimaData(1:trainCount,9)));
tic
prior = [0.4 0.6];
Mdl = fitcnb(trainData,trainLabels,...
    'PredictorNames',{  '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)'}, 'Prior',prior);
toc

%% Test
testData = pimaData(trainCount+1:end,1:8);
```

```
prediction = predict(Mdl, testData);

%% PlotConfusion
target = pimaData(trainCount+1:end,9).';
output = zeros(1, length(prediction));
output(prediction == 'D') = 1;
plotconfusion(target, output);

%% Plot Scatter Graph
figure
gscatter(testData(:,8),testData(:,3),prediction);
xlabel('Age (years)')
ylabel('Blood pressure (mm Hg)')
hold off


%% compare loss in two cnb models
defaultPriorMdl = Mdl;
defaultPriorMdl.Prior = [0.5, 0.5];
defaultCVMdl = crossval(defaultPriorMdl);
defaultLoss = kfoldLoss(defaultCVMdl);
CVMdl = crossval(Mdl);
Loss = kfoldLoss(CVMdl);

disp(strcat('defaultLoss: ', num2str(defaultLoss)));
disp(strcat('Loss: ', num2str(Loss)));
```

KNN Classifier:

```
tic;

% extracting test data from the data set---- 40% roughly (40% of 768)-----%
sample = csvread('pima-indians-diabetes.data',0,0,[0,0,306,7]);

% extracting training data from the data set---- 60% roughly (60% of 768)-----%
train = csvread('pima-indians-diabetes.data',307,0,[307,0,767,7]);

% Group vector, containg class of the train data----- %
group = csvread('pima-indians-diabetes.data',307,8,[307,8,767,8]);

% Vector, containg class of the sample/test data----- %
resultSample = csvread('pima-indians-diabetes.data',0,8,[0,8,306,8]);

% Apllying KNN classifier %
class = knnclassify(sample, train, group,7,'euclidean','nearest');

% Transpose of the resultSample vector %
resultSampleT = resultSample.';

%Transpose of the output vector of the classifier result%
classT = class.';

% Extracting dimension of the classifier out in a vector%
d = size(class);

n = d(1,1);

i=1;
```

```
count = 0;

% Comparing the the 2 vectors %
while i < n
    if class(i,1) == resultSample(i,1)
         count = count +1 ;
    end
  i = i+1;
end

accuracy = (count/n)*100;

% Plotting the confusion matrix%
plotconfusion(resultSampleT, classT);

toc;
```