MINOR PROJECT
ON
" HEART DISEASE PREDICTION USING MACHINE LEARNING "

Degree of Bachelor of Technology
in
Computer Engineering (COE)
Under the supervision of
Ms. Indu Singh
(Assistant Professor, CSE)

By:
Chetan Gupta (DTU/2K15/A1/013)
Puneet Gupta (DTU/2K15/A1/049)
Siddharth Nayyar (DTU/2K15/A1/050)

To:
Department of Computer Science and Engineering
Delhi Technological University
(Formerly Delhi College of Engineering)

TABLE OF CONTENTS

DECLARATION

We hereby certify that the work which is presented in the Minor Project entitles "**Heart Disease Prediction Using Machine Learning**" in fulfilment of the requirement for the award of the Degree of Bachelor of Technology and submitted to the Department of Computer Engineering, Delhi Technological University(Formerly Delhi College Of Engineering), New Delhi is an authentic record of my own, carried out during a period from February 2016 to June 2016, under the supervision of **Ms. Indu Singh (Assistant Professor, CSE Department)**.

The matter presented in this report has not been submitted by me for the award of any other degree of this or any other Institute/University.

Chetan Gupta (DTU/2K15/A1/013)

Puneet Gupta (DTU/2K15/A1/049)

Siddharth Nayyar (DTU/2K15/A1/050)

ACKNOWLEDGEMENT

Firstly, we express our heartiest gratitude towards the authorities who gave us a chance to explore the intricacies of various aspects of **Machine Learning**.

We are grateful to **Dr. O.P. Verma, HOD** (Department of Computer Science and Engineering), Delhi Technological University, New Delhi and all other faculty members of our department for their astute guidance throughout the project.

We would also sincerely thank our esteemed mentor, **Ms. Indu Singh**, who lent a huge helping hand in the process of making this project with her valuable guidance and blessings.

In the end, we would thank our families for their extended support throughout the project.

Chetan Gupta (DTU/2K15/A1/013)

Puneet Gupta (DTU/2K15/A1/049)

Siddharth Nayyar (DTU/2K15/A1/050)

CERTIFICATE

This is to certify that this project titled "**Heart Disease Prediction Using Machine Learning**" submitted by Chetan Gupta (DTU/2K15/A1/013), Puneet Gupta (DTU/2K15/A1/049), Siddharth Nayyar (DTU/2K15/A1/050) in partial fulfilment for the requirements for the award of Bachelor of Technology Degree in Computer Engineering (COE) at Delhi Technological University is an authentic work carried out by the students under my supervision and guidance.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other university or institute for the award of any degree or diploma.

Ms. Indu Singh
(Assistant Professor)
Department of Computer Science and Engineering
Delhi Technological University
Delhi – 110042

ABSTRACT

Classifying data is one of the most common task in Machine learning. Machine learning provides one of the main features for extracting knowledge from large databases from enterprises operational databases. Machine Learning in Medical Health Care is an emerging field of very high importance for providing prognosis and a deeper understanding of medical data. Most machine learning methods depend on a set of features that define the behaviour of the learning algorithm and directly or indirectly influence the performance as well as the complexity of resulting models.

Heart disease is the leading cause of death all over the world for the past 10 years. There have been several machine learning techniques used for the diagnosis of heart disease in the past. **Neural Network** and **Logistic Regression** are some of the few machine learning techniques used in the diagnosis of heart disease showing some amount of success.

We investigate different algorithms such as Neural Network, K-Nearest Neighbors, Naive Bayes and Logistic Regression along with hybrid techniques involving the above used algorithms for the diagnosis of heart disease.

The system has been implemented in Python platform and trained using benchmark dataset from UCI machine learning repository. The system is possibly expandable for the new dataset.

RELATED LITERATURE SURVEY

Over the past years, a lot of work and research has gone into better and accurate models for the Heart Disease Dataset. The work by Nahar J. et al. (2013) **[1]** gives a knowledge driven approach. Initially Logistic Regression was used by Dr. Robert Detrano to obtain 77% accuracy (Detrano, 1989 **[2]**). Newton Cheung utilized Naive Bayes algorithms and reached the classification accuracies of 81.48%, (Cheung, 2001 **[3]**).

Palaniappan and Awang investigated comparing different data mining techniques in the diagnosis of heart disease patients. These techniques involved naïve bayes, decision tree, and neural network. The results showed that the naïve bayes could achieve the best accuracy in the diagnosis of heart disease patients **[4]**.
K-means clustering **[5]** is one of the most popular clustering techniques; however initial centroid selection is a critical issue that strongly affects its results. This paper investigates applying different methods of initial centroid selection such as range, inlier, outlier, random attribute values, and random row methods for k-means clustering technique in the diagnosis of heart disease patients.

Indira **[6]** using Probabilistic artificial neural networks BF network. This which is a class of radial basis function RB method is useful for automatic pattern recognition, nonlinear mapping and estimation of probabilities of class membership and likelihood ratios. The data used in the experiments were taken from Cleveland heart disease database with total 576 records and 13 medical attributes. The best accuracy performance reached 94.60%.
K-Nearest-Neighbour is one of the most widely used data mining techniques in classification problems **[7]**. Its simplicity and relatively high convergence speed make it a popular choice. However a main disadvantage of KNN classifiers is the large memory requirement needed to store the whole sample. When the sample is large, response time on a sequential computer is also large **[8]**.

Our study has utilized Python and machine learning supporting libraries like scikit-learn, numpy, pandas and matplotlib. In the case of medical data diagnosis, many researchers have used a 10-fold cross validation on the total data and reported the result for disease detection, while other researchers have not used this method for heart disease prediction. For our work, we have used both test-train split idea along with cross-validation for optimal parameters selection.

INTRODUCTION

The aim of the project is to do a **“comparative study of different Machine Learning Algorithms and design an algorithm with higher accuracy”**.

Description of the problem:

Nowadays people work on computers for hours and hours, they don’t have time to take care of themselves. Due to unhealthy lifestyle, hectic schedules and consumption of junk food, the health of people is being severely affected.With the medical parameters of enough number of patients and the non-patients, it is possible to predict how likely it is for a person to be suffering from a heart disease.

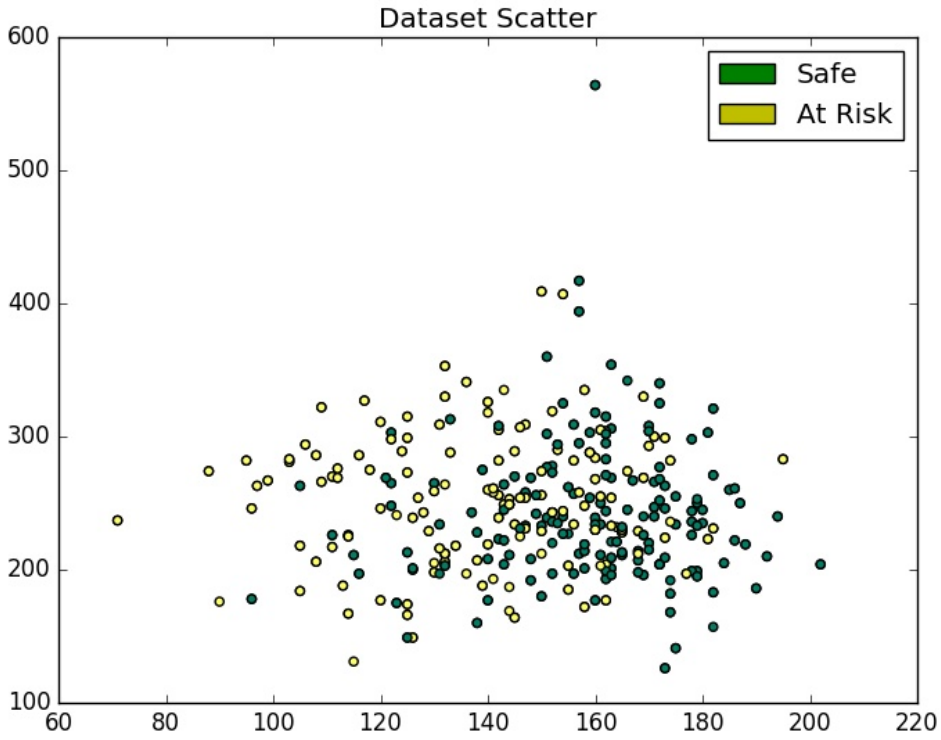
The challenge is thus to predict which of the tests are likely to turn positive on the heart disease detection test and which ones negative.

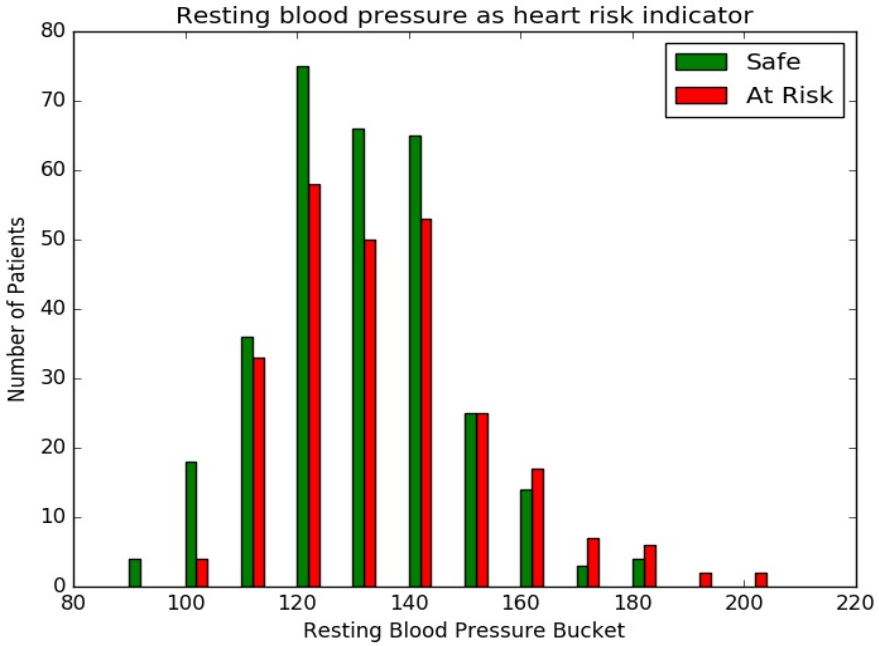
Dataset Description (UCI Heart Disease Dataset(Cleveland and Statlog))**:**

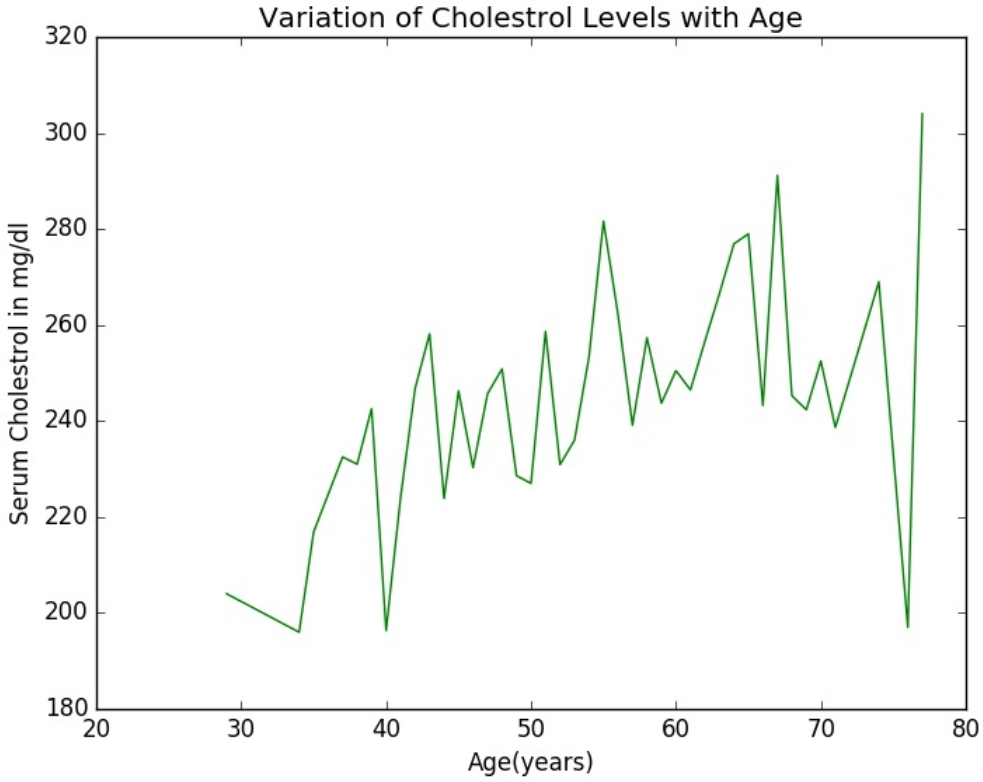
SNO.	Attribute	Description	Values
1	Age	Age in Years	Continuous
2	Gender	Male or Female	0=male, 1=female
3	Cp	Chest Pain Variety	4 = typical type 3 = typical type angina 2 = non-angina pain 1 = asymptotic
4	Thest bps	Sleeping Blood Force	Continuous
5	Chloe	Serum Cholesterol	Continuous
6	Rest Ecg	Resting ECG	5 = normal 4 = having ST_T wave abnormal 3 = left ventricular hypertrophy
7	Fbs	Fasting Blood Sugar	1 >= 120 mg/dl 0 <= 120 mg/dl
8	Thalach	Greatest heart speed reached	Continuous
9	Ex ang	Exercise induced angina	1 = no 2 = yes
10	Old peak	ST despair induced by apply virtual to relax	Continuous
11	Slope	Slope of the height effect ST section	3 = un sloping 2 = flat

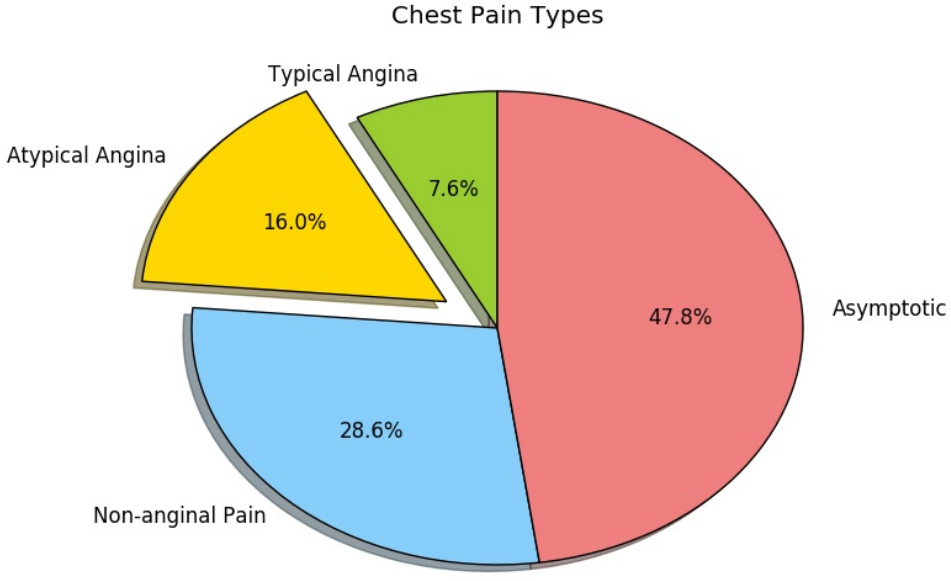
			1 = downsloping
12	Ca	Number of key vessels painted by fluoroscopy	1-4 value
13	Thal	Desert Category	2 = normal 4 = fixed 5 = reversible defect

GRAPHICAL REPRESENTATION






Variation of Cholestrol Levels with Age
Serum Cholestrol in mg/dl
Age(years)
320
300
280
260
240
220
200
180
20
30
40
50
60
70
80


Chest Pain Types
Typical Angina
Atypical Angina
Asymptotic
Non-anginal Pain
7.6%
16.0%
47.8%
28.6%

MACHINE LEARNING ALGORITHMS SELECTION AND ASSESSMENT

CLASSIFICATION PROBLEM

In machine learning, classification is the problem of identifying to which of a set of categories a new observation belongs, on the basis of a training set of data containing observations whose category membership is known.

Classification is a supervised learning problem i.e. problems in which correctly classified tuples are given and algorithm learns from these tuples. An algorithm that implements classification, especially in a concrete implementation, is known as a classifier. The term "classifier" sometimes also refers to the mathematical function, implemented by a classification algorithm, that maps input data to a category.

Binary and Multiclass Classification

Classification can be thought of as two separate problems - binary classification and multiclass classification. In binary classification, a better understood task, only two classes are involved, whereas multiclass classification involves assigning an object to one of several classes. Since many classification methods have been developed specifically for binary classification, multiclass classification often requires the combined use of multiple binary classifiers.

Comparing classification methods

Classification methods can be evaluated and compared according to the following criteria:

1) **Accuracy**: The accuracy of a classifier refers to its ability to correctly predict the label of unlabelled or previously unseen data or tuple. Higher the accuracy better will be the algorithm.
2) **Speed**: Speed of a classifier refers to how much classification time does the classification algorithm requires to complete the classification procedure.
3) **Robustness**: Robustness of a classifier is its ability to make correct classifications when the given data is noisy and has outliers.
4) **Scalability**: Scalability of a classifier is its ability to classify efficiently when large amounts of data is given.
5) **Interpretability**: This refers to the level of understanding and insight that is provided by the classifier.

ALGORITHMS

Logistic Regression

Logistic regression, despite its name, is a linear model for classification rather than regression. Logistic regression is also known in the literature as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier. In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function.

The implementation of logistic regression in scikit-learn can be accessed from class LogisticRegression. This implementation can fit a multiclass (one-vs-rest) logistic regression with optional L2 or L1 regularization.

As an optimization problem, binary class L2 penalized logistic regression minimizes the following cost function:

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1).$$

Similarly, L1 regularized logistic regression solves the following optimization problem:

$$\min_{w,c} \|w\|_1 + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1).$$

Definition of the logistic function:

An explanation of logistic regression can begin with an explanation of the standard logistic function. The logistic function is useful because it can take an input with any value from negative to positive infinity, whereas the output always takes values between zero and one and hence is interpretable as a probability. The logistic function is defined as follows:

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

K-means Clustering

The K-Means algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares. This algorithm requires the number of clusters to be specified. It scales well to large number of samples and has been used across a large range of application areas in many different fields.
The k-means algorithm divides a set of N samples X into K disjoint clusters C, each described by the mean μ_j of the samples in the cluster. The means are commonly called the cluster “centroids”; note that they are not, in general, points from X, although they live in the same space. The K-means algorithm aims to choose centroids that minimise the *inertia*, or within-cluster sum of squared criterion:

$$\sum_{i=0}^{n} \min_{\mu_j \in C} (||x_j - \mu_i||^2)$$

Inertia, or the within-cluster sum of squares criterion, can be recognized as a measure of how internally coherent clusters are. It suffers from various drawbacks:

- Inertia makes the assumption that clusters are convex and isotropic, which is not always the case. It responds poorly to elongated clusters, or manifolds with irregular shapes.

- Inertia is not a normalized metric: we just know that lower values are better and zero is optimal. But in very high-dimensional spaces, Euclidean distances tend to become inflated (this is an instance of the so-called “curse of dimensionality”). Running a dimensionality reduction algorithm such as PCA prior to k-means clustering can alleviate this problem and speed up the computations.

K-means is often referred to as Lloyd’s algorithm. In basic terms, the algorithm has three steps. The first step chooses the initial centroids, with the most basic method being to choose k samples from the dataset X. After initialization, K-means consists of looping between the two other steps. The first step assigns each sample to its nearest centroid. The second step creates new centroids by taking the mean value of all of the samples assigned to each previous centroid. The difference between the old and the new centroids are computed and the algorithm repeats these last two steps until this value is less than a threshold. In other words, it repeats until the centroids do not move significantly.

Given enough time, K-means will always converge, however this may be to a local minimum. This is highly dependent on the initialization of the centroids. As a result, the computation is often done several times, with different initializations of the centroids.

Naive Bayes

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of independence between every pair of features. Given a class variable y and a dependent feature vector x_1 through x_n, Bayes' theorem states the following relationship:

$$P(y \mid x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots x_n \mid y)}{P(x_1, \dots, x_n)}$$

Using the naive independence assumption that

$$P(x_i|y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i|y),$$

for all i, this relationship is simplified to

$$P(y \mid x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \dots, x_n)}$$

Since $P(x_1, \dots, x_n)$ is constant given the input, we can use the following classification rule:

$$P(y \mid x_1, \dots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

$$\Downarrow$$

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y),$$

and we can use Maximum A Posteriori (MAP) estimation to estimate $P(y)$ and $P(x_i \mid y)$; the former is then the relative frequency of class y in the training set.
The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of $P(x_i \mid y)$.

In spite of their apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering. They require a small amount of training data to estimate the necessary parameters.

Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods. The decoupling of the class conditional feature distributions means that each

distribution can be independently estimated as a one dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality.

K-Nearest Neighbours

In pattern recognition, the k-Nearest Neighbors algorithm (or k-NN for short) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms.
Both for classification and regression, it can be useful to assign weight to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of 1/d, where d is the distance to the neighbor.
The neighbors are taken from a set of objects for which the class (for k-NN classification) or the object property value (for k-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.
A shortcoming of the k-NN algorithm is that it is sensitive to the local structure of the data. The algorithm is not to be confused with k-means, another popular machine learning technique.

Neural Network

Machine Learning implements feed-forward artificial neural networks or, more particularly, multi-layer perceptrons (MLP), the most commonly used type of neural networks. MLP consists of the input layer, output layer, and one or more hidden layers. Each layer of MLP includes one or more neurons directionally linked with the neurons from the previous and the next layer.

All the neurons in MLP are similar. Each of them has several input links and several output links. The values retrieved from the previous layer are summed up with certain weights, individual for each neuron, plus the bias term. The sum is transformed using the activation function f that may be also different for different neurons.

In other words, given the outputs x_j of the layer n , the outputs y_i of the layer $n+1$ are computed as:

$$u_i = \sum_j (w_{i,j}^{n+1} * x_j) + w_{i,bias}^{n+1}$$

$$y_i = f(u_i)$$

Different activation functions may be used. Machine Learning implements three standard functions:

- Identity function : $f(x) = x$
- Symmetrical sigmoid : $f(x) = \beta * (1 - e^{-\alpha x})/(1 + e^{-\alpha x}$), which is the default choice for MLP.
- Gaussian function : $f(x) = \beta e^{-\alpha x * x}$, which is not completely supported at the moment.

In Machine Learning, all the neurons have the same activation functions, with the same free parameters (α, β) that are specified by user and are not altered by the training algorithms.

To compute the network, you need to know all the weights $w_{i,j}^{n+1)}$. The weights are computed by the training algorithm. The algorithm takes a training set, multiple input vectors with the corresponding output vectors, and iteratively adjusts the weights to enable the network to give the desired response to the provided input vectors.

The larger the network size is, the more the potential network flexibility is. The error on the training set could be made arbitrarily small. But at the same time the learned network also “learns” the noise present in the training set, so the error on the test set usually starts increasing after the network size reaches a limit. Besides, the larger networks are trained much longer than the smaller ones, so it is reasonable to pre-process the data and train a smaller network on only essential features.

Another MLP feature is an inability to handle categorical data as is. However, there is a workaround. If a certain feature in the input or output (in case of n -class classifier for $n > 2$) layer is categorical and can take $M > 2$ different values, it makes sense to represent it as a binary tuple of M elements, where the i -th element is 1 if and only if the feature is equal to the i -th value out of M possible. It increases the size of the input/output layer but speeds up the training algorithm convergence and at the same time enables “fuzzy” values of such variables, that is, a tuple of probabilities instead of a fixed value. Machine Learning implements two algorithms for training MLPs. The first algorithm is a classical random sequential back-propagation algorithm. The second (default) one is a batch RPROP algorithm.

Overfitting and Regularization:

Overfitting is a major problem in neural networks. This is especially true in modern networks, which often have very large numbers of weights and biases. To train effectively, we need a way of detecting when overfitting is going on, so we don't overtrain. And we'd like to have techniques for reducing the effects of overfitting.

One method of combating overfitting is called regularization. Regularization modifies the objective function that we minimize by adding additional terms that penalize large weights. In other words, we change the objective function so that it becomes Error+λf(θ), where f(θ) grows larger as the components of θgrow larger and λ is the regularization strength (a hyper-parameter for the learning algorithm). The value we choose for λ determines how much we want to protect against overfitting. A λ=0 implies that we do not take any measures against the possibility of overfitting. If λ is too large, then our model will prioritize keeping θ as small as possible over trying to find the parameter values that perform well on our training set. As a result, choosing λ is a very important task and can require some trial and error.

Fuzzy K-Nearest Neighbours

The basis of the algorithm is to assign membership as a function of the vector's distance from its K-nearest neighbors and those neighbors' memberships in the possible classes. The fuzzy algorithm is similar to the crisp version in the sense that it must also search the labeled sample set for the K-nearest neighbors. Beyond obtaining these K samples, the procedures differ considerably.
While the fuzzy K-nearest neighbor procedure is also a classification algorithm the form of its results differ from the crisp version. The fuzzy K-nearest neighbor algorithm assigns class membership to a sample vector rather than assigning the vector to a particular class. The advantage is that no arbitrary assignments are made by the algorithm. In addition, the vector's membership values should provide a level of assurance to accompany the resultant classification.

K Means Clustering with Naive Bayes Classifier

We initially implemented K Means individually. Then we worked on hybrid by combining K Means with naïve bayes. K Means is used to group together similar data. Then naïve bayes was implemented on each cluster and model was made. For each new test case, it was first determined to which cluster it belongs. Then the naïve bayes model for that particular cluster was used to make prediction for the given test case. K Means was used with the hope that grouping together similar data will help in increasing accuracy of naïve bayes algorithm. Here we had a tradeoff between time of computation and accuracy but additional gained accuracy was preferred. For this algorithm, we first discretized the data as naïve bayes requires data which is in discrete form. We couldn’t implement Gaussian naïve bayes as data distribution was not Gaussian and still using the algo would have resulted in poor accuracy. We had two choices for data discretization, equal width discretization and equal frequency discretization. Equal width discretization resulted in better performance.

CROSS VALIDATION

Cross-validation, sometimes called rotation estimation, is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. In a prediction problem, a model is usually given a dataset of known data on which training is run (training dataset), and a dataset of unknown data (or first seen data) against which the model is tested (testing dataset). The goal of cross validation is to define a dataset to "test" the model in the training phase (i.e., the validation dataset), in order to limit problems like overfitting, give an insight on how the model will generalize to an independent dataset (i.e., an unknown dataset, for instance from a real problem), etc.

In k-fold cross-validation, the original sample is randomly partitioned into k equal sized subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k − 1 subsamples are used as training data. The cross-validation process is then repeated k times (the folds), with each of the k subsamples used exactly once as the validation data. The k results from the folds can then be averaged to produce a single estimation. The advantage of this method over repeated random sub-sampling (see below) is that all observations are used for both training and validation, and each observation is used for validation exactly once. 10-fold cross-validation is commonly used but in general k remains an unfixed parameter. In stratified k-fold cross-validation, the folds are selected so that the mean response value is approximately equal in all the folds. In the case of a dichotomous classification, this means that each fold contains roughly the same proportions of the two types of class labels.

AlGORITHM IMPLEMENTATION

Graphical Representations

Importing required python modules

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
```

The following libraries have been used :

- **Pandas** : pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.
- **Numpy** : NumPy is the fundamental package for scientific computing with Python.
- **Matplotlib** : matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments .

Retrieving the Dataset

In [ ]:

```
data = pd.read_csv('heart.csv', header=None)
df = pd.DataFrame(data) #data frame
y = df.iloc[:, 13]
y = y-1
```

1. Dataset is imported.
2. The imported dataset is converted into a pandas DataFrame.
3. Attributes(x) and Labels(y) are extracted

Plotting the Dataset

In [ ]:

```
def chol_age():
    x = df.iloc[:, 0:5]
    x = x.drop(x.columns[1:4], axis=1)
    chol_avgs = x.groupby(0, sort=True).mean()
    ages = (chol_avgs[4].index.values)
    avgs = (chol_avgs[4].values)
    plt.plot(ages,avgs,'g-')
    plt.title('Variation of Cholestrol Levels with Age')
    plt.xlabel('Age(years)')
    plt.ylabel('Serum Cholestrol in mg/dl')
```

Plotting the variation of cholestrol levels with Age.

In [ ]:

```
def heart_atrack_heart_rate_bp():
    x = df.iloc[:, 0:14]
    x[14] = np.round(df[3], -1)

    x_dis = x[x[13] == 2]
    bp_set_dis = x_dis.groupby(14, sort=True)
    nums_dis = (bp_set_dis.count()[0]).index.values
    bps_dis = (bp_set_dis.count()[0]).values
    bar2 = plt.bar(nums_dis+2, bps_dis, color='r', width=2)

    x_nor = x[x[13] == 1]
    bp_set_nor = x_nor.groupby(14, sort=True)
    nums_nor = (bp_set_nor.count()[0]).index.values
    bps_nor = (bp_set_nor.count()[0]).values
    bar1 = plt.bar(nums_nor, bps_nor, color='g', width=2)

    plt.title('Resting blood pressure as heart risk indicator')
    plt.xlabel('Resting Blood Pressure Bucket')
    plt.ylabel('Number of Patients')
    plt.legend((bar1[0], bar2[0]), ('Safe', 'At Risk'))
```

Showing the resting blood pressure as a heart disease risk indicator.

In [ ]:

```
def pie_chart_chest_pain():
    x = df.iloc[:, 0:3]
    sets = x.groupby(2).count()
    fin_lab = ['Typical Angina', 'Atypical Angina', 'Non-anginal Pain', 'Asymptotic
    values = (sets[0].values)
    plt.pie(values,
        labels=fin_lab,
        colors=['yellowgreen', 'gold', 'lightskyblue', 'lightcoral'],
        explode = [0,0.2,0,0],
        shadow=True,
        autopct='%1.1f%%',
        startangle=90)
    plt.title('Chest Pain Types')
```

A pie chart of chest pain types.

In [ ]:

```
def scatter_chart():
    x = df.iloc[:, 0:13]
    sc = plt.scatter(x[7],x[4], c=y, cmap='summer')
    plt.title('Dataset Scatter')
    classes = ['Safe', 'At Risk']
    class_colours = ['g','y']
    recs = []
    for i in range(0,len(class_colours)):
        recs.append(mpatches.Rectangle((0,0),1,1,fc=class_colours[i]))
    plt.legend(recs, classes)
```

The dataser scatter showing the safe and at risk records.

Logistic Regression

Importing required python modules

In [1]:

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import train_test_split
from sklearn import metrics
from sklearn.cross_validation import cross_val_score
import numpy as np
import pandas as pd
```

The following libraries have been used :

- **Pandas** : pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.
- **Numpy** : NumPy is the fundamental package for scientific computing with Python.
- **Matplotlib** : matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments .
- **Sklearn** : It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Retrieving the Dataset

In [ ]:

```
data = pd.read_csv('heart.csv', header=None)

df = pd.DataFrame(data)

x = df.iloc[:, 0:13]
y = df.iloc[:, 13]
y = y-1
```

1. Dataset is imported.
2. The imported dataset is converted into a pandas DataFrame.
3. Attributes(x) and Labels(y) are extracted

In [ ]:

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4)
```

Train/Test Split is 0.4

Plotting the Dataset

In [ ]:

```
fig = plt.figure()
ax1 = fig.add_subplot(1,2,1)
ax1.scatter(x[3],x[4], c=y)
ax1.set_title("Original Data")
```

Matplotlib is used to plot the loaded pandas DataFrame.

Learning from the Dataset

In [ ]:

```
c = 3
model = LogisticRegression(C = c)


scores = cross_val_score(model, x, y, scoring='accuracy', cv=10)
print ("10-Fold Accuracy : ", scores.mean()*100)

model = model.fit(x_train,y_train)
print ("Testing Accuracy : ",model.score(x_test, y_test)*100)
predicted = model.predict(x)
```

Here **model** is an instance of LogisticRegression method from sklearn.linear_model. **c** is the regularisation constatnt(1/lambda). 10 Fold Cross Validation is used to verify the results.

In [ ]:

```
ax2 = fig.add_subplot(1,2,2)
ax2.scatter(x[3],x[4], c=predicted)
ax2.set_title("Logistic Regression")
```

The learned data is plotted.

In [ ]:

```
cm = metrics.confusion_matrix(y, predicted)
print (cm/len(y))
print (metrics.classification_report(y, predicted))

plt.show()
```

Compute confusion matrix to evaluate the accuracy of a classification and build a text report showing the main classification metrics.

Kmeans Clustering

Importing required python modules

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.cross_validation import train_test_split
from sklearn.preprocessing import normalize,scale
from sklearn.cross_validation import cross_val_score
from sklearn import metrics
```

The following libraries have been used :

- **Pandas** : pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.
- **Numpy** : NumPy is the fundamental package for scientific computing with Python.
- **Matplotlib** : matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments .
- **Sklearn** : It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Retrieving the dataset

In [ ]:

```
data = pd.read_csv('heart.csv', header=None)
df = pd.DataFrame(data)

x = df.iloc[:, 0:5]
x = x.drop(x.columns[1:3], axis=1)
x = pd.DataFrame(scale(x))

y = df.iloc[:, 13]
y = y-1
```

1. Dataset is imported.
2. The imported dataset is converted into a pandas DataFrame.
3. Attributes(x) and labels(y) are extracted.

Plotting the Dataset

In [ ]:

```
fig = plt.figure()

ax1 = fig.add_subplot(1,2,1)
ax1.scatter(x[1],x[2], c=y)
ax1.set_title("Original Data")
```

Matplotlib is used to plot the loaded pandas DataFrame.

Learning from the data:

In [ ]:

```
clusters = 2

model = KMeans(init='k-means++', n_clusters=clusters,
n_init=10,random_state=100)

scores = cross_val_score(model, x, y, scoring='accuracy', cv=10)
print ("10-Fold Accuracy : ", scores.mean()*100)

model.fit(x)

predicts = model.predict(x)
print ("Accuracy(Total) = ", count(predicts == np.array(y))/(len(y)*1.0) *100)
centroids = model.cluster_centers_
```

Here **model** is an instance of KMeans method from sklearn.cluster. The number of clusters to form are taken as 2. The initial cluster centers for k-mean clustering are selected in a smart way to speed up convergence. 10 Fold Cross Validation is used to verify the results.

In [ ]:

```
ax1.scatter(centroids[:, 1], centroids[:, 2],
            marker='x', s=169, linewidths=3,
            color='b', zorder=10)

ax2 = fig.add_subplot(1,2,2)
ax2.set_title("KMeans Clustering")
ax2.scatter(x[1],x[2], c=predicts)
ax2.scatter(centroids[:, 1], centroids[:, 2],
            marker='x', s=169, linewidths=3,
            color='b', zorder=10)
```

The learned cluster centroids are then used for prediction and to plot the clustered dataset.

In [ ]:

```
cm = metrics.confusion_matrix(y, predicts)
print (cm/len(y))
print (metrics.classification_report(y, predicts))

plt.show()
```

Compute confusion matrix to evaluate the accuracy of a classification and build a text report showing the main classification metrics.

Naive Bayes Classification

Importing required python modules

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
from sklearn.cross_validation import train_test_split
from sklearn.preprocessing import normalize,scale
from sklearn.cross_validation import cross_val_score
from sklearn import metrics
```

The following libraries have been used :

- **Pandas** : pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.
- **Numpy** : NumPy is the fundamental package for scientific computing with Python.
- **Matplotlib** : matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments .
- **Sklearn** : It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Retrieving the dataset

In [ ]:

```
data = pd.read_csv('heart.csv', header=None)
df = pd.DataFrame(data)

x = df.iloc[:, 0:13]
y = df.iloc[:, 13]
y = y-1

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4)
```

1. Dataset is imported.
2. The imported dataset is converted into a pandas DataFrame.
3. Attributes(x) and labels(y) are extracted.
4. Train/Test split is 0.4

Plotting the Dataset

In [ ]:

```
fig = plt.figure()
ax1 = fig.add_subplot(1,2,1)
ax1.scatter(x[3],x[4], c=y)
ax1.set_title("Original Data")
```

Matplotlib is used to plot the loaded pandas DataFrame.

Learning from the data:

In [ ]:

```
model = MultinomialNB()

scores = cross_val_score(model, x, y, scoring='accuracy', cv=10)
print ("10-Fold Accuracy : ", scores.mean()*100)

model.fit(x_train,y_train)
predicts = model.predict(x)
```

Here **model** is an instance of MultinomialNB method from sklearn.naive_bayes. Additive (Laplace/Lidstone) smoothing parameter is 1. Class prior properties are learned. 10 Fold Cross Validation is used to verify the results.

In [ ]:

```
ax2 = fig.add_subplot(1,2,2)
ax2.scatter(x[3],x[4], c=predicts)
ax2.set_title("Naive Bayes")
```

The learned cluster centroids are then used for prediction and to plot the clustered dataset.

In [ ]:

```
cm = metrics.confusion_matrix(y, predicts)
print (cm/len(x))
print (metrics.classification_report(y, predicts))

plt.show()
```

Compute confusion matrix to evaluate the accuracy of a classification and build a text report showing the main classification metrics.

KNN

Importing required python modules

In [ ]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cross_validation import train_test_split
from sklearn import metrics
from sklearn.preprocessing import normalize,scale
from sklearn.cross_validation import cross_val_score
import numpy as np
import pandas as pd
```

The following libraries have been used :

- **Pandas** : pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.
- **Numpy** : NumPy is the fundamental package for scientific computing with Python.
- **Matplotlib** : matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments .
- **Sklearn** : It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Retrieving the dataset

In [ ]:

```
data = pd.read_csv('heart.csv', header=None)

df = pd.DataFrame(data)


x = df.iloc[:, 0:5]
x = x.drop(x.columns[1:3], axis=1)
x = pd.DataFrame(scale(x))

y = df.iloc[:, 13]
y = y-1
```

1. Dataset is imported.
2. The imported dataset is converted into a pandas DataFrame.
3. Attributes(x) and labels(y) are extracted.

In [ ]:

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4)
```

Train/Test split is 0.4

Plotting the dataset

In [ ]:

```
fig = plt.figure()
ax1 = fig.add_subplot(1,2,1)
ax1.scatter(x[1],x[2], c=y)
ax1.set_title("Original Data")
```

Matplotlib is used to plot the loaded pandas DataFrame.

Learning from the data

In [ ]:

```
model = KNeighborsClassifier(n_neighbors=5)


scores = cross_val_score(model, x, y, scoring='accuracy', cv=10)
print ("10-Fold Accuracy : ", scores.mean()*100)

model.fit(x_train,y_train)
print ("Testing Accuracy : ",model.score(x_test, y_test)*100)
predicted = model.predict(x)
```

Here **model** is an instance of KNeighborsClassifier method from sklearn.neighbors. 10 Fold Cross Validation is used to verify the results.

In [ ]:

```
ax2 = fig.add_subplot(1,2,2)
ax2.scatter(x[1],x[2], c=predicted)
ax2.set_title("KNearestNeighbours")
```

The learned data is plotted.

In [ ]:

```
cm = metrics.confusion_matrix(y, predicted)
print (cm/len(y))
print (metrics.classification_report(y, predicted))


plt.show()
```

Compute confusion matrix to evaluate the accuracy of a classification and build a text report showing the main classification metrics.

Fuzzy K-Nearest Neighbours

Importing required python modules

In [ ]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cross_validation import train_test_split
from sklearn import metrics
from sklearn.preprocessing import normalize,scale
from sklearn.cross_validation import cross_val_score
import numpy as np
import pandas as pd
```

The following libraries have been used :

- **Pandas** : pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.
- **Numpy** : NumPy is the fundamental package for scientific computing with Python.
- **Matplotlib** : matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments .
- **Sklearn** : It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Retrieving the dataset

In [ ]:

```
data = pd.read_csv('heart.csv', header=None)

df = pd.DataFrame(data)

x = df.iloc[:, 0:5]
x = x.drop(x.columns[1:3], axis=1)
x = pd.DataFrame(scale(x))

y = df.iloc[:, 13]
y = y-1
```

1. Dataset is imported.
2. The imported dataset is converted into a pandas DataFrame.
3. Attributes(x) and labels(y) are extracted.

In [ ]:

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4)
```

Train/Test split is 0.4

Plotting the dataset

In [ ]:

```
fig = plt.figure()
ax1 = fig.add_subplot(1,2,1)
ax1.scatter(x[1],x[2], c=y)
ax1.set_title("Original Data")
```

Matplotlib is used to plot the loaded pandas DataFrame.

Learning from the data

In [ ]:

```
model = KNeighborsClassifier(n_neighbors=5, weights='distance')

scores = cross_val_score(model, x, y, scoring='accuracy', cv=10)
print ("10-Fold Accuracy : ", scores.mean()*100)

model.fit(x_train,y_train)
print ("Testing Accuracy : ",model.score(x_test, y_test)*100)
predicted = model.predict(x)
```

Here **model** is an instance of KNeighborsClassifier method from sklearn.neighbors. The number of neighbors used is 5 and the weights function predcits weight points by the inverse of their distance, i.e closer neighbors of a query point will have a greater influence than neighbors which are further away. 10 Fold Cross Validation is used to verify the results.

In [ ]:

```
ax2 = fig.add_subplot(1,2,2)
ax2.scatter(x[1],x[2], c=predicted)
ax2.set_title("Fuzzy KNearestNeighbours")
```

The learned data is plotted.

In [ ]:

```
cm = metrics.confusion_matrix(y, predicted)
print (cm/len(y))
print (metrics.classification_report(y, predicted))


plt.show()
```

Compute confusion matrix to evaluate the accuracy of a classification and build a text report showing the main classification metrics.

Kmeans Clustering with Naive Bayes Classifier

Importing required python modules

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.cross_validation import train_test_split
from sklearn.preprocessing import scale
from sklearn.cross_validation import cross_val_score
from sklearn import metrics
from sklearn.naive_bayes import MultinomialNB
from sklearn.cross_validation import KFold
```

The following libraries have been used :

- **Pandas** : pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.
- **Numpy** : NumPy is the fundamental package for scientific computing with Python.
- **Matplotlib** : matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments .
- **Sklearn** : It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Retrieving the dataset

In [ ]:

```
data = pd.read_csv('heart.csv', header=None)
df = pd.DataFrame(data)
```

1. Dataset is imported.
2. The imported dataset is converted into a pandas DataFrame.

Plotting the Dataset

In [ ]:

```
fig = plt.figure()

ax1 = fig.add_subplot(1,2,1)
ax1.scatter(x[1],x[2], c=y)
ax1.set_title("Original Data")
FP = 0
FN = 0
TN = 0
TP = 0
```

Matplotlib is used to plot the loaded pandas DataFrame.

Learning from the data:

In [ ]:

```
def nbkmh(train_index, test_index):
```

This function is used to run the hybrid classifier.

In [ ]:

```
    x_kmeans = df.iloc[:, 0:5]
    x_kmeans = x_kmeans.drop(x_kmeans.columns[1:3], axis=1)
    x_kmeans = pd.DataFrame(scale(x_kmeans))

    x_naive = df.iloc[:, 0:13]
    y = df.iloc[:, 13]
    y = y-1

    y_train = pd.Series(y.iloc[train_index])
    y_test = pd.Series(y.iloc[test_index])

    x_train_kmeans = x_kmeans.iloc[train_index, :]
    x_test_kmeans = x_kmeans.iloc[test_index, :]

    x_train_naive = x_naive.iloc[train_index, :]
    x_test_naive = x_naive.iloc[test_index, :]
```

Labels and attributes are extracted from the dataset for Kmeans and Naive Bayes respectively. Kmeans Clustering uses only the continous attributes.

In [ ]:

```
    clusters = 5
    model_kmeans = KMeans(init='k-means++', n_clusters=clusters, n_init=10,random_s
    model_kmeans.fit(x_train_kmeans)
    kmean_predictions = model_kmeans.predict(x_train_kmeans)
```

Kmeans clustering is run on the dataset to cluster the data into 5 clusters. The initial cluster centers for k-mean clustering are selected in a smart way to speed up convergence.

In [ ]:

```
    x = [pd.DataFrame() for ii in range(0,clusters)]
    y = [pd.Series() for ii in range(0,clusters)]
    iterators = zip(kmean_predictions,range(len(x_train_kmeans)))
    for kmean_prediction,i in iterators:
        row_x =  x_train_naive.iloc[i, :]
        row_y = pd.Series(y_train.iloc[i])
        index = int(kmean_prediction)
        x[index] = x[index].append(row_x, ignore_index=True)
        y[index] = y[index].append(row_y)
```

Attributes(x) and labels(y) are then grouped according to the cluster defined by the Kmeans Clustering.

In [ ]:

```
    clstr_n = [MultinomialNB(alpha=2,fit_prior=True) for ii in range(0,clusters)]
    for i in range(0,clusters):
        clstr_n[i].fit(x[i], y[i])
```

Naive Bayes Classifier is then run on each cluster individually. Additive (Laplace/Lidstone) smoothing parameter is set as 2. Class prior probabilities are learned.

In [ ]:

```
    predicts = []
    c=0
    for i in range(len(x_test_kmeans)):
        prediction = model_kmeans.predict(
                    x_test_kmeans.iloc[i, :].reshape(1,-1))
        prediction = int(prediction)
        pred_naive = clstr_n[prediction].predict(
                    x_test_naive.iloc[i, :].reshape(1,-1))
        predicts.append(pred_naive)
        if pred_naive == y_test.iloc[i]:
            c+=1
    print ((c*100.0)/len(x_test_kmeans))
```

Accuracies are predicted on the test set using the hybrid classifier.

In [ ]:

```
    predicts = np.array(predicts)
    cm = metrics.confusion_matrix(y_test, predicts)/len(y_test)
    # print (cm)
    global FP
    global FN
    global TN
    global TP
    FP += cm[0][0]
    FN += cm[1][0]
    TN += cm[0][1]
    TP += cm[1][1]
    return ((c*100.0)/len(x_test_kmeans))
```

Compute confusion matrix to evaluate the accuracy of a classification and build a text report showing the main classification metrics.

In [ ]:

```
def main():
    scores = []
    kf = KFold(n=df.shape[0], n_folds=10)
    for (train_index,test_index),i in zip(kf,range(0,10)):
        print("Iteration " + str(i+1) + " : ")
        scores.append(nbkmh(train_index, test_index))
    print("\n 10 Fold Accuracy",np.array(scores).mean())
    print("FP", FP*10)
    print("FN", FN*10)
    print("TN", TN*10)
    print("TP", TP*10)

if __name__ == '__main__':
    main()
```

This is the function that is used to call the function nbkmh() and run 10 Fold Cross Validation.

Neural Network

Importing required python modules

In [ ]:

```
import numpy as np
from scipy import optimize
from sklearn.preprocessing import scale
from sklearn import metrics
```

The following libraries have been used :

- **Numpy** : NumPy is the fundamental package for scientific computing with Python.
- **Scipy** : Scipy is a collection of numerical algorithms and domain-specific toolboxes, including signal processing, optimization, statistics and much more.
- **Sklearn** : It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Learning from the data

In [ ]:

```
def featureNormalize(z):
    return scale(z)
def sigmoid(z):
    r = 1.0 / (1.0 + np.exp(-z))
    return r
def sigmoidGrad(z):
    r = sigmoid(z)
    r = r * (1.0 - r)
    return r
def randomizeTheta(l, epsilon):
    return ((np.random.random((l, 1)) * 2 * epsilon) - epsilon)
```

In [ ]:

```
def KFoldDiv(X, y, m, n, K):
    sz = int(np.ceil(m / K))
    if n == 1:
        X_train = X[sz:, :]
        X_test = X[:sz, :]
        y_train = y[sz:]
        y_test = y[:sz]
    elif n == K:
        X_train = X[:((n-1)*sz), :]
        X_test = X[((n-1)*sz):, :]
        y_train = y[:((n-1)*sz)]
        y_test = y[((n-1)*sz):]
    else:
        X_train = np.vstack((X[:((n-1)*sz), :], X[(n*sz):, :]))
        X_test = X[((n-1)*sz):(n*sz), :]
        y_train = np.vstack((y[:((n-1)*sz)], y[(n*sz):]))
        y_test = y[((n-1)*sz):(n*sz)]
    return (X_train, y_train, X_test, y_test)
```

Auxiliary Functions:

- **featureNormalize** : Scales the attributes of the dataset.
- **sigmoid** : Computes sigmoid function on the given data.
- **sigmoidGrad** : Computes derivative of sigmoid function on the given data.
- **randomizeTheta** : Generates a set of random weights for the purpose of initialization of weights.
- **KFoldDiv** : It is a function which divides the dataset into train and test datasets, based on the fold number for cross validation.

In [ ]:

```
def nnCostFunc(Theta, input_layer_size, hidden_layer_size, num_labels, X, y, lmbda)
    Theta1, Theta2 = np.split(Theta, [hidden_layer_size * (input_layer_size+1)])
    Theta1 = np.reshape(Theta1, (hidden_layer_size, input_layer_size+1))
    Theta2 = np.reshape(Theta2, (num_labels, hidden_layer_size+1))
    m = X.shape[0]
    y = (y == np.array([(i+1) for i in range(num_labels)])).astype(int)

    a1 = np.hstack((np.ones((m, 1)), X))
    z2 = np.dot(a1, Theta1.T)
    a2 = np.hstack((np.ones((m, 1)), sigmoid(z2)))
    h = sigmoid(np.dot(a2, Theta2.T))

    cost = ((lmbda/2)*(np.sum(Theta1[:, 1:] ** 2) +
            np.sum(Theta2[:, 1:] ** 2)) -
            np.sum((y * np.log(h)) +
            ((1-y) * np.log(1-h)))) / m
    return cost
```

nnCostFunc: It computes the cost function for neural networks with regularization, which is given by,

$$Cost(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\ln\left((h_\theta(x^{(i)}))_k\right) - (1-y_k^{(i)})\ln\left(1-(h_\theta(x^{(i)}))_k\right)\right] + \frac{\lambda}{2m}\left[\sum_{i=1}(\theta_i)^2\right]$$

The neural network has 3 layers – an input layer, a hidden layer and an output layer. It uses forward propagation to compute $(h_\theta(x^{(i)}))_k$, the activation (output value) of the k-th output unit and θ represents the weights. The code works for any number of input units, hidden units and outputs units.

In [ ]:

```
def nnGrad(Theta, input_layer_size, hidden_layer_size, num_labels, X, y, lmbda):
    Theta1, Theta2 = np.split(Theta, [hidden_layer_size * (input_layer_size+1)])
    Theta1 = np.reshape(Theta1, (hidden_layer_size, input_layer_size+1))
    Theta2 = np.reshape(Theta2, (num_labels, hidden_layer_size+1))
    m = X.shape[0]
    y = (y == np.array([(i+1) for i in range(num_labels)])).astype(int)

    a1 = np.hstack((np.ones((m, 1)), X))
    z2 = np.dot(a1, Theta1.T)
    a2 = np.hstack((np.ones((m, 1)), sigmoid(z2)))
    h = sigmoid(np.dot(a2, Theta2.T))

    delta_3 = h - y
    delta_2 = np.dot(delta_3, Theta2[:, 1:]) * sigmoidGrad(z2)
    Theta2_grad = (np.dot(delta_3.T, a2) +
                   (lmbda * np.hstack((np.zeros((Theta2.shape[0], 1)),
                                       Theta2[:, 1:])))) / m
    Theta1_grad = (np.dot(delta_2.T, a1) +
                   (lmbda * np.hstack((np.zeros((Theta1.shape[0], 1)),
                                       Theta1[:, 1:])))) / m

    grad = np.hstack((Theta1_grad.flatten(), Theta2_grad.flatten()))
    return grad
```

nnGrad: It computes the gradient(also called partial derivative) of the cost function with respect to all weights in the neural network. The gradient helps in optimizing the weights in order to minimize the value of the cost function.

In [ ]:

```
K = 10
lmbda = 0.03
epsilon = 0.12

input_layer_size = 13
hidden_layer_size = 20
num_labels = 2
```

Initialisation of relevant parameters.

In [ ]:

```
X = np.genfromtxt('heart.csv', delimiter=',')
m, n = X.shape
n -= 1

y = X[:, n].astype(int).reshape((m, 1))
X = featureNormalize(X[:, :n])
foldAcc = np.ndarray((K, 1))
```

Import the dataset and extract labels and attributes from it.

In [ ]:

```
FP = 0
FN = 0
TN = 0
TP = 0
for i in range(K):
    X_train, y_train, X_test, y_test = KFoldDiv(X, y, m, i+1, K)

    initTheta = randomizeTheta((hidden_layer_size * (input_layer_size+1)) +
                               (num_labels * (hidden_layer_size+1)), epsilon)
    Theta = optimize.fmin_bfgs(nnCostFunc, initTheta, fprime=nnGrad,
                               args=(input_layer_size,
                                     hidden_layer_size,
                                     num_labels, X_train,
                                     y_train,
                                     lmbda),
                               maxiter=3000)
    Theta1, Theta2 = np.split(Theta, [hidden_layer_size * (input_layer_size+1)])
    Theta1 = np.reshape(Theta1, (hidden_layer_size, input_layer_size+1))
    Theta2 = np.reshape(Theta2, (num_labels, hidden_layer_size+1))

    h1 = sigmoid(np.dot(np.hstack((np.ones((X_test.shape[0], 1)), X_test)), Theta1.
    h2 = sigmoid(np.dot(np.hstack((np.ones((h1.shape[0], 1)), h1)), Theta2.T))
    predicted = h2.argmax(1) + 1
    predicted = predicted.reshape((predicted.shape[0], 1))
    foldAcc[i] = np.mean((predicted == y_test).astype(float)) * 100

    cm = (metrics.confusion_matrix(y_test, predicted))/len(y_test)

    FP += cm[0][0]
    FN += cm[1][0]
    TN += cm[0][1]
    TP += cm[1][1]

    print('Test Set Accuracy for %dth fold: %f\n' % (i+1, foldAcc[i]))
meanAcc = np.mean(foldAcc)
print('\nAverage Accuracy: ', meanAcc)
print("")
print(FP)
print(FN)
print(TN)
print(TP)
```

The above written code is used to run 10 Fold Cross Validation on the Neural Network and display the Model Accuracy and the Confusion Matrix and related metrics.

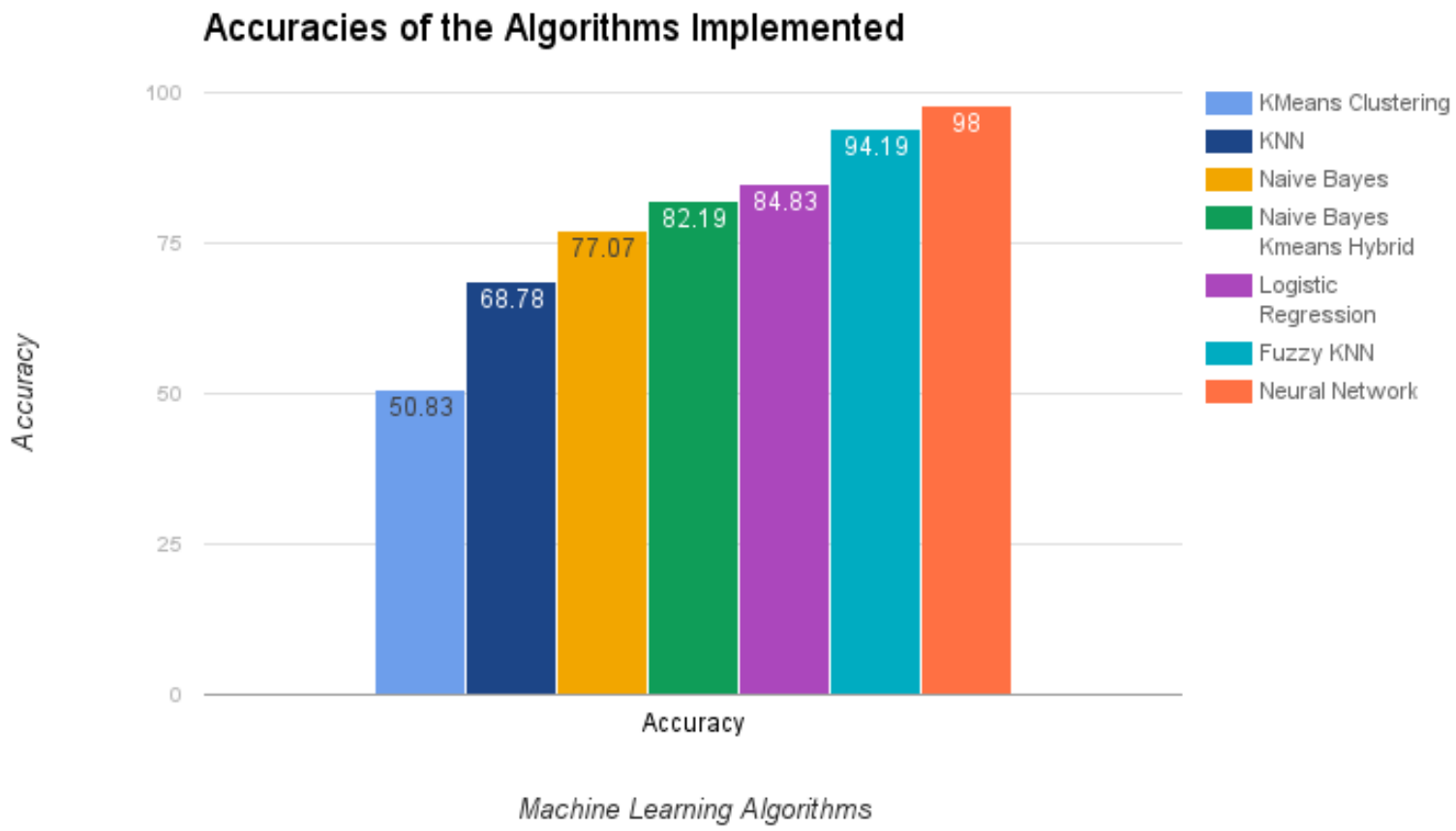
fmin_bfgs function from **Scipy** library is used to optimize the weights in order to minimize the cost, using the BFGS algorithm.

Parameters:

- f : callable f(x,*args), *Objective function to be minimized.*
- x0 : ndarray, *Initial guess.*
- fprime : callable f'(x,*args), *Gradient of f.*
- args : tuple, *Extra arguments passed to f and fprime.*

RESULT

The results of the various machine learning algorithms are as follows:



CONFUSION MATRICES

1. K Means Clustering

	True	False
Positive	30.8	14.6
Negative	31.5	23.1

2. K Nearest Neighbours

	True	False
Positive	31.2	14.1
Negative	39.5	15.2

3. Naive Bayes

	True	False
Positive	33.1	11.1
Negative	43.5	12.3

4. K Means Clustering with Naive Bayes

	True	False
Positive	35.1	7.7
Negative	47	10.2

5. Logistic Regression

	True	False
Positive	37.7	6.9
Negative	47.9	7.5

6. Fuzzy K Nearest Neighbours

	True	False
Positive	39.8	2.6
Negative	52.2	5.4

7. Neural Network

	True	False
Positive	43.9	0.2
Negative	54.5	1.4

ACCURACY COMPARISONS WITH LATEST RESEARCH PAPERS

The following table illustrates the accuracy of our Hybrid Algorithm with its counterparts.

Author/Year	**Technique**	**Accuracy**
Muhammad Fathurachman/2014	Extreme Learning Machine	84.00%
Divyansh Khanna, Rohan Sahu, Veeky Baths, Bharat Deshpande/2015	Logistic Regression Generalized regression Neural Network	82.80% 89.00%
Kemal Polat, Seral Sahan, Salih Gunes/2005	Fuzzy-AIRS–k-nn based system	87.00%
V Krishnaiah/2015	Fuzzy KNN	84.00%
Our Work	K Means Clustering with Naive Bayes **Fuzzy KNN** **BP-Neural Network**	82.19% 94.19% 98.00%

CONCLUSION

The project undertook the study of various algorithms that include Neural Network, Naive Bayes, K-Nearest Neighbors and Logistic Regression that can be effectively implemented in Python to predict the heart attacks and then to compare the best method of prediction that is to be used for diagnosis of heart disease. Along with basic algorithms, the hybrid algorithms generated perform a much better analysis and give better performance than various research papers which have already been cited.

From the experimental result and analysis, it can be concluded that the performance of Neural Network and Fuzzy-KNN when compared with the performance of K-Means Clustering, KNN, Logistic Regression, etc. tends to be better. Regularisation of the model which helps in increasing the overall accuracy of the algorithm by preventing overfitting, has been implemented in our approach. Fuzzy-KNN assures that an incorrectly classified sample will not have a membership in any class close to one, while a correctly classified sample does possess a membership in the correct class close to one.

As is known that heart disease has a complex pathology. In the future, models of heart disease detection system can be further developed, therefore the development of our models still need advice and suggestions from an expert and adding some attributes of patient data to determine the intensity of heart disease whether heart disease is already at the level of chronic or not yet.

The project can be used as an important tool for doctors and health experts to predict certain critical cases in the practice and used to advise the patient accordingly. The model from the classification will be able to answer more complex queries in the prediction of heart attack diseases.

REFERENCES

[1] J. Nahar, “Computational intelligence for heart disease diagnosis: A medical knowledge driven approach,” Expert Systems with Applications, pp. 96-104, 2013.

[2] R. Detrano, A. Janosi, W. Steinbrunn, M. Pfisterer, J. Schmid, S. Sandhu et al., “International application of a new probability algorithm for the diagnosis of coronary artery disease,” American Journal of Cardiology, vol. 6, pp. 304-310, 1989.

[3] N. Cheung, “Machine learning techniques for medical analysis,” B.Sc. Thesis, School of Information Technology and Electrical Engineering, University of Queensland, 2001.

[4] Palaniappan, S. and R. Awang, Web-Based Heart Disease Decision Support System using Data Mining Classification Modeling Techniques. Proceedings of iiWAS, 2007.

[5] Bramer, M., Principles of data mining. 2007: Springer.

[6] I. S. F. Dessai, “Intelligent Heart Disease Prediction System Using Probabilistic Neural Network,” Karnataka, 2013.

[7] F. Moreno-Seco, L. Mico, and J.A. Oncina, "Modification of the LAESA Algorithm for Approximated k-NN Classification," Pattern Recognition Letters, pp. 47–53, 2003.

[8] E. Alpaydin, Voting over Multiple Condensed Nearest Neighbors. Artificial Intelligence Review, pp. 115–132. 1997.

[9] D. Kriesel. A Brief Introduction to Neural Networks. [Online]. Available: http://www.dkriesel.com/en/science/neural_networks

[10] C. M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, 1995.

[11] James M. Keller, A Fuzzy k-Nearest Neighbor Algorithm, IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, VOL. SMC-15, NO. 4, JULY/AUGUST 1985

[12] Muhammad Fathurachman, Heart Disease Diagnosis using Extreme Learning Based Neural Networks, 2014 International Conference of Advanced Informatics: Concept, Theory and Application (ICAICTA)

[13] V Krishnaiah, Diagnosis of Heart Disease Patients Using Fuzzy Classification Technique, Computer and Communications Technologies (ICCCT), 2014 International Conference

[14] Shouman, Mai, Tim Turner, and Rob Stocker. "Applying k-nearest neighbour in diagnosing heart disease patients." International Journal of Information and Education Technology 2.3 (2012): 220.