SAVITRIBAI PHULE PUNE UNIVERSITY

A PRELIMINARY PROJECT REPORT ON

"FOREST TYPE PREDICTION USING CLASSIFIERS "

SUBMITTED TOWARDS THE PARTIAL FULFILMENT OF THE REQUIREMENTS OF

LABORATORY PRACTICE - III (MACHINE LEARNING)

Academic Year: 2019-20

By:

Aditi Deshpande (BECOA125)

Anand Kulkarni (BECOA160)

Under The Guidance of

Prof. Alka Londhe

DEPARTMENT OF COMPUTER ENGINEERING,

PCET'S PIMPRI CHINCHWAD COLLEGE OF ENGINEERING

Sector No. 26, Prabhakaran, Nigdi,
Pune - 411044

PCET'S PIMPRI CHINCHWAD COLLEGE OF ENGINEERING
Sector No. 26, Pradhikaran, Nigdi,
Pune - 411044

DEPARTMENT OF COMPUTER ENGINEERING

Certificate

This is to certify that the Mini Project report entitled

"FOREST TYPE PREDICTION USING CLASSIFIERS"

Submitted By

Aditi Deshpande BECOA125
Anand Kulkarni BECOA160

is approved by **Prof. Alka Londhe** for submission. It is certified further that, to the best of my knowledge, the report represents work carried out by my students as the partial fulfillment for BE. Computer Engineering (Semester II) Laboratory-III Work (Machine Learning) as prescribed by the Savitribai Phule Pune University for the academic year 2019-20.

Prof. Alka Londhe
(Mini Project Guide)

Place: Pune
Date:

ABSTRACT

Deforestration is considered as an important part in the economy . To calculate the actual count of the Deforestration and make the analysis most of the forests parts are unknown.Thus there should be the appropriate prediction system.

Forest Prediction using Classifiers is the system which is highly scalable product for finding the type of the unknown type of the forest region. It offers various classification techniques such as SVM,Decision Tree,Logistic Regression,Naïve Bayes.
Forest Prediction using Classifiers solution is bundled with an excellent predicting capability with largest accuracy.

INDEX

List of Figures & Tables

Chapter 1: Introduction

a) Problem Statement

Different tree types have varying growth rates. Deforestation and tree logging are compensated for by the planting of new trees. However, species and grow rate variations are not currently included in deforestation calculations. Thus, the aim is to predict the type of forest by studying features in dataset.

b) Project Idea

To apply Machine Learning Classification techniques to determine forest types and help in estimation for real deforestation rates.

c) Motivation

It was general observation that some of the forest area are not counted in deforestation calculations of the region since the type of the forests were not known. Also there is need to manage the resources to its fullest. Thus we came up with an idea to develop a software application to manage challenges faced by the deforestration administration team for keeping track of all the region in the forest prediction and classification to over come the problem of unkown forest region.

d) Scope

Forest type classification using classification is a software application for forest administration department. It is designed to predict the unknown forest region using various classification techniques which are widely used . It uses various geographical features.
The application is made for streamlining all the activities associated with the forest type prediction under a common umbrella and cut down on the time and effort required.
The system is ideal for all small/big forest regions having large number of observations.

e) Literature Survey/ Requirement Analysis

The traditional method that is used in prediction had use of complex mathematics this also involved detail analysis and stastistics .
The current proposed system predicts the required output in a feasiable and less with less manual computaions and high accuracy.
Requirment analysis involves various geographical features appropriate considerations for model building

Chapter 2: Project Design

a) H/W , S/W , resources, requirements & their detail explanation

The requirements section of hardware includes minimum of 3 GB hard disk and 4 GB RAM with 2 GHz or higher speed. The primary requirements include a memory of 4GB for the application. As this is an desktop application so we are not enabling or installing any hardware components for user interface. It's not an embedded system.

- Processor - Intel I3
- Speed - 1.5 Ghz and Above
- RAM - 4 GB (min)
- Hard Disk - 10 GB(max)
- Key Board - Standard Windows Keyboard
- Mouse - Two or Three Button Mouse

This is the software configuration in which the project was shaped. The programming language used, tools used, etc are described here.

Operating System : Windows
Tool : Python,Jupyter notebook

b) Dataset Design

This data set contains training and testing data from a remote sensing study which mapped different forest types based on their spectral characteristics at visible-to-near infrared wavelengths, using ASTER satellite imagery. The output (forest type map) can be used to identify and/or quantify the ecosystem services (e.g. carbon storage, erosion protection) provided by the forest.

Attribute Information:

Class: 's' ('Sugi' forest), 'h' ('Hinoki' forest), 'd' ('Mixed deciduous' forest), 'o' ('Other' non-forest land) b1 - b9: ASTER image bands containing spectral information in the green, red, and near infrared wavelengths for three dates (Sept. 26, 2010; March 19, 2011; May 08, 2011. pred_minus_obs_S_b1 - pred_minus_obs_S_b9: Predicted spectral values (based on spatial interpolation) minus actual spectral values for the 's' class (b1-b9). pred_minus_obs_H_b1 - pred_minus_obs_H_b9: Predicted spectral values (based on spatial interpolation) minus actual spectral values for the 'h' class (b1-b9).

c) Hours estimation

The estimated time for model building and model execution with appropriate prediction along with accuracy measures is 10minutes maximum.

Chapter 3: Module Description

a) Block diagram with explanation of each module

1)Support Vector Machine(SVM)

Support Vector Machine(SVM) is a supervised machine learning which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes.The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.

Various Terminologies associated are:

Gamma :

The gamma parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’. In other words, with low gamma, points far away from plausible seperation line are considered in calculation for the seperation line. Where as high gamma means the points close to plausible line are considered in calculation.

Linear Kernel SVM

The dot-product is called the kernel and can be re-written as:

K(x, xi) = sum(x * xi)

The kernel defines the similarity or a distance measure between new data and the support vectors. The dot product is the similarity measure used for linear SVM or a linear kernel because the distance is a linear combination of the inputs

Polynomial Kernel SVM

Instead of the dot-product, we can use a polynomial kernel, for example:

K(x,xi) = 1 + sum(x * xi)^d

Where the degree of the polynomial must be specified by hand to the learning algorithm. When d=1 this is the same as the linear kernel. The polynomial kernel allows for curved lines in the input space.

Radial Kernel SVM

Finally, we can also have a more complex radial kernel. For example:

K(x,xi) = exp(-gamma * sum((x – xi^2))

Where gamma is a parameter that must be specified to the learning algorithm. A good default value for gamma is 0.1, where gamma is often 0 < gamma < 1. The radial kernel is very local and can create complex regions within the feature space, like closed polygons in two-dimensional space.

2) Logistic Regression

. Logistic regression has become an important tool in the discipline of machine learning . The approach allows an algorithm being used in a machine learning application to classify incoming data based on historical data. As more relevant data comes in, the algorithm should get better at predicting classifications within data sets.

Logistic regression is a statistical method for analysing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes).

In logistic regression, the dependent variable is binary or dichotomous, i.e. it only contains data coded as 1 (TRUE, success, pregnant, etc.) or 0 (FALSE, failure, non-pregnant, etc.).

The goal of logistic regression is to find the best fitting (yet biologically reasonable) model to describe the relationship between the dichotomous characteristic of interest (dependent variable = response or outcome variable) and a set of independent (predictor or explanatory) variables. Logistic regression generates the coefficients (and its standard errors and significance levels) of a formula to predict a *logit transformation* of the probability of presence of the characteristic of interest:

$$logit(p) = b_0 + b_1X_1 + b_2X_2 + b_3X_3 + \ldots + b_kX_k$$

where p is the probability of presence of the characteristic of interest. The logit transformation is defined as the logged odds:

$$odds = \frac{p}{1-p} = \frac{probability\ of\ presence\ of\ characteristic}{probability\ of\ absence\ of\ characteristic}$$

and

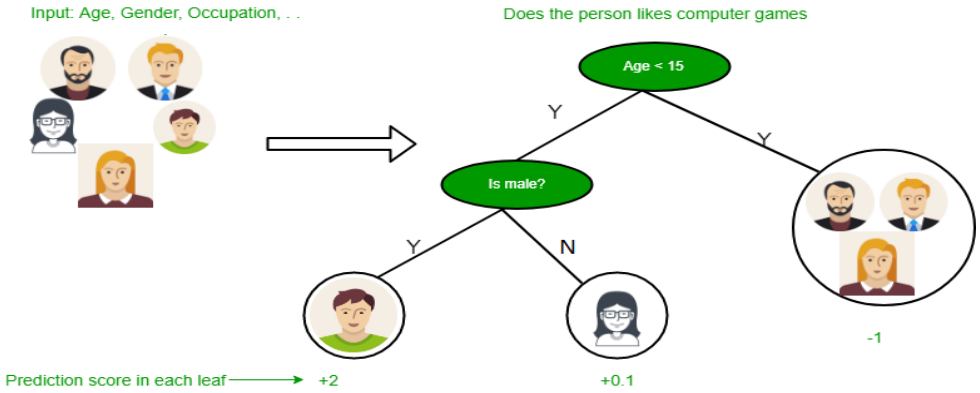
$$logit(p) = \ln\left(\frac{p}{1-p}\right)$$

Rather than choosing parameters that minimize the sum of squared errors (like in ordinary regression), estimation in logistic regression chooses parameters that maximize the likelihood of observing the sample values

It estimates discrete values **(**Binary values like 0/1, yes/no, true/false) based on a given set of independent variables(s). It predicts the probability of occurrence of an event by fitting data to a *logit function***.** Hence, it is also known as ***logit regression***. The values obtained would always lie within 0 and 1 since it predicts the probability.

3)Decision Trees

Decision Trees (DTs**)** are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.



In Decision Tree the major challenge is to identification of the attribute for the root node in each level. This process is known as attribute selection. We have two popular attribute selection measures:

1. Information Gain
2. Gini Index

1. Information Gain

When we use a node in a decision tree to partition the training instances into smaller subsets the entropy changes. Information gain is a measure of this change in entropy.

Entropy

Entropy is the measure of uncertainty of a random variable, it characterizes the impurity of an arbitrary collection of examples. The higher the entropy more the information content.

2. Gini Index

Gini Index is a metric to measure how often a randomly chosen element would be incorrectly identified.
It means an attribute with lower Gini index should be preferred.
Sklearn supports “Gini” criteria for Gini Index and by default, it takes “gini” value

The strengths of decision tree methods are:

- Decision trees are able to generate understandable rules.
- Decision trees perform classification without requiring much computation.

- Decision trees are able to handle both continuous and categorical variables.
- Decision trees provide a clear indication of which fields are most important for prediction or classification.

The weaknesses of decision tree methods :

- Decision trees are less appropriate for estimation tasks where the goal is to predict the value of a continuous attribute.
- Decision trees are prone to errors in classification problems with many class and relatively small number of training examples.
- Decision tree can be computationally expensive to train. The process of growing a decision tree is computationally expensive. At each node, each candidate splitting field must be sorted before its best split can be found.

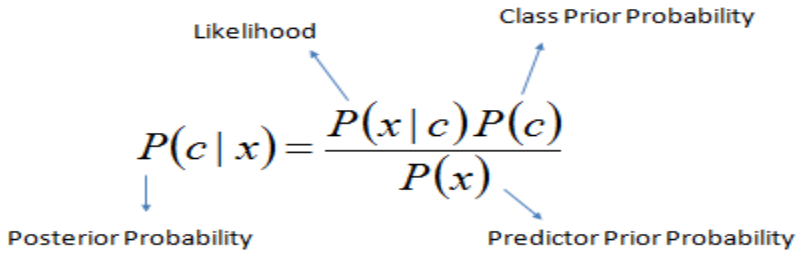
4)Naive Bayes Classifier

This is a classification technique based on an assumption of independence between predictors or what's known as *Bayes' theorem*. In simple terms, a ***Naive Bayes classifier*** assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, a Naive Bayes Classifier would consider all of these properties to independently contribute to the probability that this fruit is an apple.

To build a Bayesian model is simple and particularly functional in case of enormous data sets. Along with simplicity, Naive Bayes is known to outperform sophisticated classification methods as well.

Bayes theorem provides a way of calculating posterior probability **P(c|x)** from **P(c)**, **P(x)** and **P(x|c)**. The expression for Posterior Probability is as follows.



$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Here,

- $P(c|x)$ is the posterior probability of *class* (*target*) given *predictor* (*attribute*).
- $P(c)$ is the prior probability of *class.*
- $P(x|c)$ is the likelihood which is the probability of *predictor* given *class.*

$P(x)$ is the prior probability of *predictor*.

Chapter 4: Results & Discussion

a) Source code

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import pandas as pd
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import precision_recall_fscore_support
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

dataset = pd.read_csv('forest.csv')
dataset.head()

Enc = LabelEncoder()
data_tf = dataset.copy()
for i in dataset.columns:
  data_tf[i]=Enc.fit_transform(dataset[i])
X = data_tf.drop(['class'], axis=1)
y = data_tf['class'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 82)
param_grid = {'C': [1, 10, 100, 1000], 'gamma': [1, 0.1, 0.01, 0.001, 0.0001]}
.
grid = GridSearchCV(SVC(), param_grid, verbose=2)
grid.fit(X_train,y_train)

grid.best_params_
grid_predictions = grid.predict(X_test)
print(confusion_matrix(y_test,grid_predictions))
```

```
print(classification_report(y_test,grid_predictions))
print(grid_predictions)
print(confusion_matrix(y_test,grid_predictions))
print(classification_report(y_test,grid_predictions))
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

svcclassifier1 = SVC(kernel = 'linear', random_state = 0)
svcclassifier1.fit(X_train, y_train)

x_pred1 = svcclassifier1.predict(X_train)
y_pred1 = svcclassifier1.predict(X_test)
print(y_pred1)

y_compare1 = np.vstack((y_test,y_pred1)).T

y_compare1[:5,:]
print('*********FOR LINEAR KERNAL*********')

ac1=accuracy_score(y_test, y_pred1)
print('\n Testing Accuracy of SVM using Linear Kernel :',ac1*100,end='%')
tac1 = accuracy_score(y_train, x_pred1)
print('\n Training Accuracy of SVM using Linear Kernal :',tac1*100,end='%')
print('\n1) Confusion Matrix:  ')
cm1 = confusion_matrix(y_test, y_pred1)
print(cm1)
print('\n2) Shape of Confusion Matrix')
a1 = cm1.shape
print(a1)
corrPred1 = 0
falsePred1 = 0
for row1 in range(a1[0]):
    for c1 in range(a1[1]):

        if row1 == c1:
            corrPred1 +=cm1[row1,c1]
        else:
            falsePred1 += cm1[row1,c1]
print('\n3) Correct predictions:  ', corrPred1)
print('\n4)False predictions: ', falsePred1)
kernelLinearAccuracy = corrPred1/(cm1.sum())
```

```
print("\n5)Precision Score:  {0:.4f}".format(precision_score(y_test,
y_pred1,average='weighted')))
print("\n6)Recall Score: {0:.4f}".format(recall_score(y_test, y_pred1, average='weighted')
print("\n7)F1 Score: {0:.4f}".format(f1_score(y_test,y_pred1,  average='weighted')
print ('\n8)Accuracy of the SVC Clasification (USING LINEAR KERNAL)is: ',
kernelLinearAccuracy*100,end='%')
vcclassifier2 = SVC(kernel = 'poly', random_state = 0)
svcclassifier2.fit(X_train, y_train)
x_pred2 = svcclassifier2.predict(X_train)
y_pred2 = svcclassifier2.predict(X_test)
print(y_pred2)
y_compare2 = np.vstack((y_test,y_pred2)).T
y_compare2[:5,:]
print('*********FOR POLYNOMIAL KERNAL*********')
ac2=accuracy_score(y_test, y_pred2)
print('\n Testing Accuracy of SVM using Polynomial Kernel :',ac2*100,end='%')
tac2 = accuracy_score(y_train, x_pred2)
print('\n Training Accuracy of SVM using Polynomial Kernal :',tac2*100,end='%')
print('\n1) Confusion Matrix:  ')
cm2 = confusion_matrix(y_test, y_pred2)
print(cm2)
print('\n2) Shape of Confusion Matrix')
a2 = cm2.shape
print(a2)
corrPred2 = 0
falsePred2 = 0
for row2 in range(a2[0]):
  for c2 in range(a2[1]):
       if row2 == c2:
          corrPred2 +=cm2[row2,c2]
       else:
          falsePred2 += cm2[row2,c2]
print('\n3) Correct predictions:  ', corrPred2)
print('\n4)False predictions: ', falsePred2)
kernelPolyAccuracy = corrPred2/(cm2.sum())
print("\n5)Precision Score:  {0:.4f}".format(precision_score(y_test,
y_pred2,average='weighted')))
print("\n6)Recall Score: {0:.4f}".format(recall_score(y_test, y_pred2, average='weighted')))
print("\n7)F1 Score: {0:.4f}".format(f1_score(y_test,y_pred2,  average='weighted')))
print ('\n8)Accuracy of the SVC Clasification (using Polynomial kernal)is: ',
kernelPolyAccuracy*100,end='%')
```

```
svcclassifier3 = SVC(kernel = 'rbf', random_state = 0)
svcclassifier3.fit(X_train, y_train)
x_pred3 = svcclassifier3.predict(X_train)
y_pred3 = svcclassifier3.predict(X_test)
print(y_pred3)
y_compare3 = np.vstack((y_test,y_pred3)).T
y_compare3[:5,:]
print('**********FOR RBF KERNAL**********')
ac3=accuracy_score(y_test, y_pred3)
print('\n Testing Accuracy of SVM using RBF Kernel :',ac3*100,end='%')
tac3 = accuracy_score(y_train, x_pred3)
print('\n Training Accuracy of SVM using RBF Kernal :',tac3*100,end='%')
print('\n1) Confusion Matrix:  ')
cm3 = confusion_matrix(y_test, y_pred3)
print(cm3)
print('\n2) Shape of Confusion Matrix')
a3 = cm3.shape
print(a3)
corrPred3 = 0
falsePred3 = 0
for row3 in range(a3[0]):
    for c3 in range(a3[1]):
        if row3 == c3:
            corrPred3 +=cm3[row3,c3]
        else:
            falsePred3 += cm3[row3,c3]
print('\n3) Correct predictions:  ', corrPred3)
print('\n4)False predictions: ', falsePred3)
kernelRbfAccuracy = corrPred3/(cm3.sum())
print("\n5)Precision Score:  {0:.4f}".format(precision_score(y_test,
y_pred3,average='weighted')))
print("\n6)Recall Score: {0:.4f}".format(recall_score(y_test, y_pred3, average='weighted')))
print("\n7)F1 Score: {0:.4f}".format(f1_score(y_test,y_pred3,  average='weighted')))
print ( 'Accuracy of the SVC Clasification (using rbf kernal) is: ', kernelRbfAccuracy*100)
svcclassifiers4 = SVC(kernel = 'sigmoid', random_state = 0)
svcclassifiers4.fit(X_train, y_train)
x_pred4 = svcclassifiers4.predict(X_train)
y_pred4 = svcclassifiers4.predict(X_test)
print(y_pred4)
y_compare4 = np.vstack((y_test,y_pred4)).T
y_compare4[:5,:]
print('**********FOR SIGMOID KERNAL**********')
ac4=accuracy_score(y_test, y_pred4)
print('\n Testing Accuracy of SVM using Sigmoid Kernel :',ac4*100,end='%')
```

```
tac4 = accuracy_score(y_train, x_pred3)
print('\n Training Accuracy of SVM using Sigmoid Kernal :',tac4*100,end='%')
print('\n1) Confusion Matrix:  ')
cm4 = confusion_matrix(y_test, y_pred4)
print(cm4)
print('\n2) Shape of Confusion Matrix')
a4 = cm4.shape
print(a4)
corrPred4 = 0
falsePred4 = 0
for row4 in range(a4[0]):
    for c4 in range(a4[1]):
        if row4 == c4:
            corrPred4 +=cm4[row4,c4]
        else:
            falsePred4 += cm3[row4,c4]
     print('\n3) Correct predictions:  ', corrPred4)
print('\n4)False predictions: ', falsePred4)
kernelSfAccuracy = corrPred4/(cm4.sum())
print("\n5)Precision Score:  {0:.4f}".format(precision_score(y_test,
y_pred4,average='weighted')))
print("\n6)Recall Score: {0:.4f}".format(recall_score(y_test, y_pred4, average='weighted')
print("\n7)F1 Score: {0:.4f}".format(f1_score(y_test,y_pred4,  average='weighted')))
print ( 'Accuracy of the SVC Clasification (using Sigmoid kernal) is: ',
kernelSfAccuracy*100)
# Decision Tree
print("\n-----------------------------------------------------")
print("\t\t\tDecision Tree")
model_DecisionTree = DecisionTreeClassifier()
model_DecisionTree.fit(X_train, y_train)
y_predict=model_DecisionTree.predict(X_test)
X_predict = model_DecisionTree.predict(X_train)
print("\nConfusion matrix for decision tree classifier : ")
print(confusion_matrix(y_test,y_predict))
ac=accuracy_score(y_test, y_predict)
print('\n Testing Accuracy for Decision Tree :',ac*100,end='%')
tac=accuracy_score(y_train, X_predict)
print('\n Training Accuracy for Decision Tree :',tac*100,end='%')
pre=precision_recall_fscore_support(y_test, y_predict)
#print(pre)
print("\n\nPrecision for Decision Tree : ",pre[0][0])
#print(pre[0][0]) #precision   (tp/(tp+fp)
print("\nRecall for Decision Tree : ",pre[1][0])
```

```
#print(pre[1][0]) #Recall     (tp/tp+fn)
print("\n-----------------------------------------------------")

# Logistic regression
print("\n-----------------------------------------------------")
print("\t\t\tLogistic Regression")
clf = LogisticRegression(random_state=0, solver='newton-cg',
multi_class='multinomial').fit(X_train, y_train)
y_pred = clf.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print('\nConfusion matrix for Logistic Regression : ')
print(cm)
#score = clf.score(y_test,y_pred)
score = clf.score(X_test,y_test)
print('\nTesting Accuracy for Logistic Regression :',score*100,end='%')
tscore = clf.score(X_train,y_train)
print('\nTraining Accuracy for Logistic Regression :',tscore*100,end='%')
pre = precision_recall_fscore_support(y_test, y_predict)
#print(pre)
print("\n\nPrecision for Logistic Regression : ",pre[0][0])
#print(pre[0][0]) #precision    (tp/(tp+fp)
print("\nRecall for Logistic Regression : ",pre[1][0])
#print(pre[1][0]) #Recall     (tp/tp+fn)
print("\n-----------------------------------------------------")
```

b) Screen shots

1) Conversion of dataset using Label Encoder

	b9	pred_minus_obs_H_b1	...	pred_minus_obs_S_b1
	67	47.70	...	-18.27
	59	47.93	...	-20.13
	57	53.09	...	-17.64
	59	52.41	...	-20.20
	56	68.54	...	-18.62

```
t LabelEncoder


(dataset[i])
```

b8	b9	pred_minus_obs_H_b1	...	\
12	20	72	...	
7	12	75	...	
7	10	112	...	
9	12	104	...	

2) Value of Hyperparameter Gamma

```
param_grid = {'C': [1, 10, 100, 1000], 'gamma': [1, 0.1, 0.01, 0.001, 0.0001]
# dictionary for values of parameters C and gamma.
grid = GridSearchCV(SVC(), param_grid, verbose=2)
grid.fit(X_train,y_train)
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
[CV] C=1, gamma=1 ....................................................
[CV] ..................................... C=1, gamma=1, total=   0.0s
[CV] C=1, gamma=1 ....................................................
[CV] ..................................... C=1, gamma=1, total=   0.0s
[CV] C=1, gamma=1 ....................................................
[CV] ..................................... C=1, gamma=1, total=   0.0s
[CV] C=1, gamma=0.1 ..................................................
[CV] ..................................... C=1, gamma=0.1, total=   0.0s
[CV] C=1, gamma=0.1 ..................................................
[CV] ..................................... C=1, gamma=0.1, total=   0.0s
[CV] C=1, gamma=0.1 ..................................................
[CV] ..................................... C=1, gamma=0.1, total=   0.0s
[CV] C=1, gamma=0.01 .................................................
[CV] .................................... C=1, gamma=0.01, total=   0.0s
[CV] C=1, gamma=0.01 .................................................
[CV] .................................... C=1, gamma=0.01, total=   0.0s
[CV] C=1, gamma=0.01 .................................................
[CV] .................................... C=1, gamma=0.01, total=   0.0s
[CV] C=1, gamma=0.001 ................................................
```

```
29]: grid.best_params_
```

```
29]: {'C': 10, 'gamma': 0.0001}
```

3) Linear Kernel Output

```
*********FOR LINEAR KERNAL*********

 Testing Accuracy of SVM using Linear Kernel : 80.0%
 Training Accuracy of SVM using Linear Kernal : 92.37288135593221%
1) Confusion Matrix:
[[14  0  0  1]
 [ 0  6  0  3]
 [ 3  0  5  1]
 [ 4  0  0 23]]

2) Shape of Confusion Matrix
(4, 4)

3) Correct predictions:   48

4)False predictions:  12

5)Precision Score:  0.8363

6)Recall Score: 0.8000

7)F1 Score: 0.7980

8)Accuracy of the SVC Clasification (USING LINEAR KERNAL)is:  80.0%
```

4) Polynomial Kernal Output

```
*********FOR POLYNOMIAL KERNAL*********

 Testing Accuracy of SVM using Polynomial Kernel : 70.0%
 Training Accuracy of SVM using Polynomial Kernal : 89.83050847457628%
1) Confusion Matrix:
[[10  0  0  5]
 [ 0  4  0  5]
 [ 5  0  3  1]
 [ 2  0  0 25]]

2) Shape of Confusion Matrix
(4, 4)

3) Correct predictions:   42

4)False predictions:  18

5)Precision Score:  0.7596

6)Recall Score: 0.7000

7)F1 Score: 0.6807

8)Accuracy of the SVC Clasification (using Polynomial kernal)is:  70.0%
```

5) Output of RBF Kernal

```
*********FOR RBF KERNAL*********

 Testing Accuracy of SVM using RBF Kernel : 81.66666666666667%
 Training Accuracy of SVM using RBF Kernal : 92.79661016949152%
1) Confusion Matrix:
[[14  0  0  1]
 [ 0  6  0  3]
 [ 4  0  5  0]
 [ 2  1  0 24]]

2) Shape of Confusion Matrix
(4, 4)

3) Correct predictions:   49

4)False predictions:  11

5)Precision Score:  0.8393

6)Recall Score: 0.8167

7)F1 Score: 0.8124
Accuracy of the SVC Clasification (using rbf kernal) is:  81.66666666666667
```

6) Output of Decision Tree

```
------------------------------------------------------------
                        Decision Tree

Confusion matrix for decision tree classifier :
[[10  1  0  4]
 [ 0  7  0  2]
 [ 2  1  5  1]
 [ 6  3  0 18]]

 Testing Accuracy for Decision Tree : 66.66666666666666%
 Training Accuracy for Decision Tree : 100.0%

Precision for Decision Tree :  0.5555555555555556

Recall for Decision Tree :  0.6666666666666666

------------------------------------------------------------
```

7)Output of Logistic Regression

```
------------------------------------------------------------
                        Logistic Regression

Confusion matrix for Logistic Regression :
[[13  1  0  1]
 [ 0  6  0  3]
 [ 2  1  5  1]
 [ 2  1  0 24]]

Testing Accuracy for Logistic Regression : 80.0%
Training Accuracy for Logistic Regression : 90.67796610169492%

Precision for Logistic Regression :  0.5555555555555556

Recall for Logistic Regression :  0.6666666666666666

------------------------------------------------------------
```

8)Output of Naïve Bayes

```
------------------------------------------------------------
                        Naive Bayes

Confusion matrix for Naive Bayes classifier :
[[11  0  1  3]
 [ 0  7  0  2]
 [ 3  0  6  0]
 [ 3  4  0 20]]

Testing Accuracy for Naive Bayes : 73.33333333333333%
Training Accuracy for Naive Bayes : 81.77966101694916%
Precision for Naive Bayes classifier :  0.6470588235294118

Recall for Naive Bayes classifier :  0.7333333333333333

------------------------------------------------------------
```

Chapter 5: Conclusion

Conclusion Table

Classifiers and Performance Parameter	SVM(Liner Kernel) in%	SVM(Polynomial Kernek)in %	SVM(RBF Kernel)in %	SVM(Sigmoid Kernel) ib %	Decision Tree in %	Logistic Regression in %	Naïve Bayes in %
Accuracy	80.00	70	81	75	68	80	73
Precision	83.63	0.75	0.83	0.78	0.58	0.58	0.64
Recall	80	0.7	0.81	0.75	0.68	0.66	0.73
F1 Score	79.89	0.68	0.81	0.73	0.67	0.79	0.74
Correct Prediction	48	42	49	45	48	42	45
False Prediction	12	18	11	11	12	18	12
Training Accuracy	92.97	70	92.79	92	100	90.67	81
Testing Accuracy	80	89	81.9	75	68.33	80.33	73

The results obtained using various Classifiers show an accuracy ranging from 68-81% . Highest accuracy among them is obtained by,SVM(using rbf kernel).