Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

CRIME PREDICTION

TEAM MEMBERS:

G.HARINISRI - 19MIA1069

MADASU DEEPIKA – 19MIA1066

SHIVANI GOKULRAM NAIDU -19MIA1006

ABSTRACT

Crime is one of the serious issues in our society. It is the most predominant aspect of our society. It is also predominant in society. So, the prevention of crime is one of the important tasks. The crime analysis should be in a systematic way. As the analysis makes it important in the detecting and prevention of crime. The analysis detects the investigating patterns and helps in the detection of trends in crime. The main of this paper is the analysis of the efficiency of the crime investigation. The model is designed for the detection of crime patterns from inferences. The inferences are collected from the crime scene and these inferences, the paper demonstrates the prediction that will happen. The dataset used in this paper is taken from the" Communities and Crime dataset" from UCI repository . There is a need for analysing the crime data to lower the crime rate. This helps the police and citizens to take necessary actions and solve the crimes faster. In this paper, data mining techniques are applied to crime data for predicting features that affect the high crime rate.

INTRODUCTION

Criminal activities across the globe have created a menace in the society. Every year large volume of criminal data is generated by the law enforcement organizations and it is a major challenge for them to analyse this data to implement decision for avoiding crimes in future. Analysing this data not only helps in recognizing a feature responsible for high crime rate but also helps in taking necessary actions for prevention of crimes. Predictive analysis is a statistical technique used to develop models that predict future events. These predictive models usually are measured using scores on which the features are extracted are taken as accuracy, precision, recall and F1 score.Data mining also involves classification with prediction that helps to get a even more better accurate results.

Classification and Prediction involves classifying a class label for data using probability equations and predicting any feature using numeric measures accordingly. 80% of data will be trained according to the given algorithms and tested on the rest of 20% data. The aim is to predict top most features with the accurate predictive model that affects the high crime rate which will eventually help police or law enforcement makers take necessary actions.

Here the algorithms that we choose are:

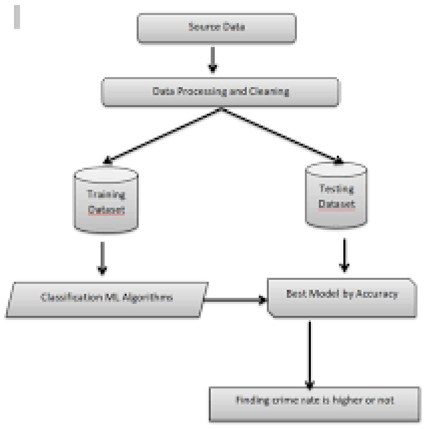
- Decision tree
- KNN
- Linear SVC
- Gaussian Naïve Bayes classification
- Polynomial SVC
- Random forest
- Gradient boosting classification

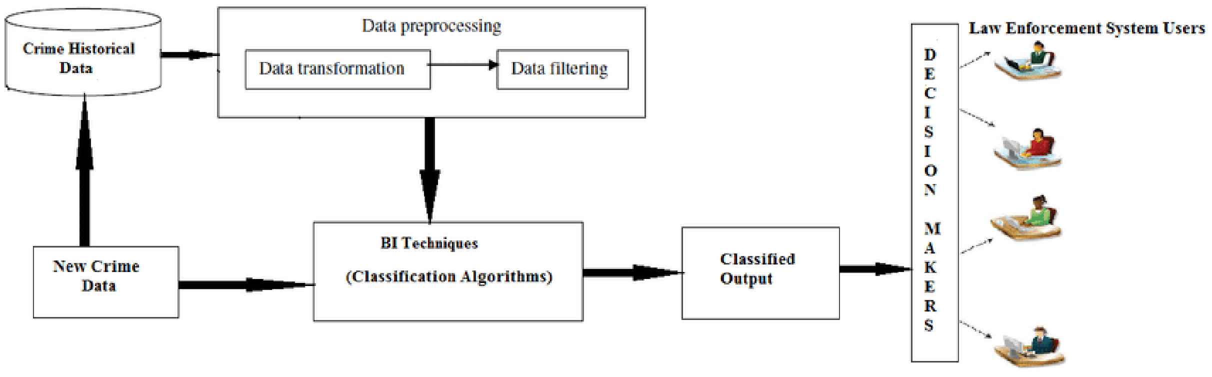
And all with cross validation method to calculate the accuracy scores that can make us choose.

DATASET

In this paper, for performing predictive analysis, the Communities and Crime dataset from UCI repository has been used which consists of crime data in Chicago, a city having highest crime rate in the United States of America. It includes features affecting crime rate like population, race, sex, immigrants etc. Many features involving the community like, such as the percent of the population considered urban, and the median family income, and involving law enforcement, such as per capita number of police officers, and percent of officers assigned to drug units are included so that algorithms that select or learn weights for attributes could be tested . The attribute or feature to be predicted is ‘Per Capita Violent Crimes’ which was pre-calculated in the data using population and the sum of crime variables considered violent crimes in the United States: murder, rape, robbery, and assault.

ARCHITECTURE DIAGRAM


Source Data
Data Processing and Cleaning
Training Dataset
Testing Dataset
Classification ML Algorithms
Best Model by Accuracy
Finding crime rate is higher or not



SOFTWARE USED

The overall system has been developed using python with support of Jupyter notebook.

But here I want to highlight a few important libraries that have been used:

Pandas: It is an open source library that provides high-performance, easy-to-use data structures and data analysis tools for the Python programming language. It stores the tabular, matrix data into rows and columns using data frames in Python which helps to process the data dynamically.

Numpy: It is a powerful package for scientific computing in Python. It can be used by creating an N-dimensional object array usually represented by np.

Scikit Learn: Most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python.

MODULES

DECISION TREE CLASSIFIER

Decision tree model is a supervised learning method of classification. The goal of this classifier [16] is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A deeper tree indicates complex decision rules and that makes a better fitted model. Entropy gives the information gain from a decision rule. Hence for this predictive model, Entropy as a criterion for splitting of branches has been used.

KNN CLASSIFIER

KNN algorithms use data and classify new data points based on similarity measures (e.g. distance function). Classification is done by a majority vote to its neighbors.

LINEAR SVC

algorithm for solving multiclass classification problems from ultra large data sets that implements an original proprietary version of a cutting plane algorithm for designing a linear support vector machine.

GAUSSIAN NB CLASSIFIER

GaussianNB module make use of Naïve Bayes theorem that are a set of supervised learning algorithms with the “naive” assumption of independence between every pair of features. If y is an attribute to be predicted and X are the attributes used for prediction.

POLYNOMIAL SVC

In machine learning, the polynomial kernel is a kernel function commonly used with support vector machines (SVMs) and other kernelized models, that represents the similarity of vectors (training samples) in a feature space over polynomials of the original variables, allowing learning of non-linear models.

RANDOM FOREST CLASSIFIER

This classifier fits n number of decision tree classifiers on various sub-samples of the dataset, controls the over-fitting of data and improves predictive accuracy by averaging. This classifier has been used to gain a good accuracy over singular decision trees obtained in 3.

GRADIENT BOOSTING CLASSIFIER

group of machine learning algorithms that combine many weak learning models together to create a strong predictive model. Decision trees are usually used when doing gradient boosting. The Gradient Boosting Classifier depends on a loss function.

SAMPLE CODE

WE REQUIRE A CLEAN DATASET IN ORDER TO CONDUCT OUR ANALYSIS (done by pre-processing)

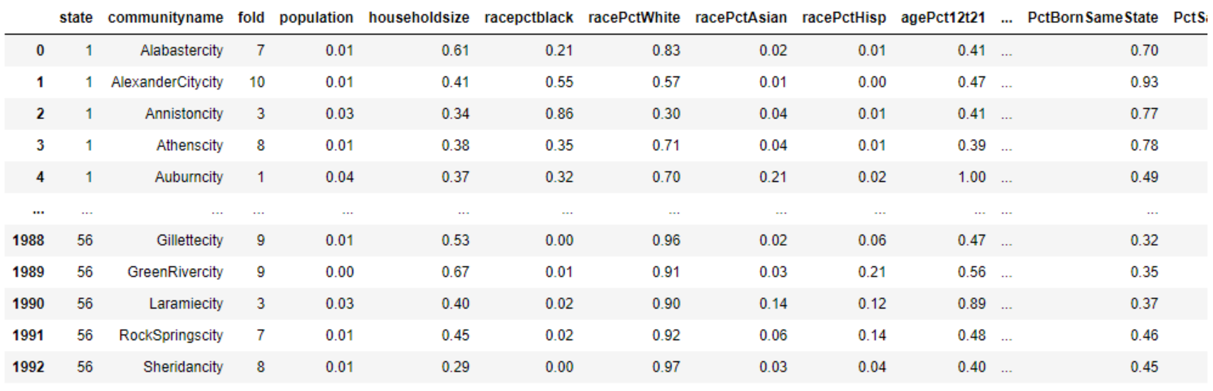
IMPORTING THE REQUIRED LIBRARIED:

```
import pandas as pd

import numpy as np

import sklearn.metrics as metrics

from sklearn.tree import DecisionTreeClassifier, export_graphviz

from sklearn import svm

from sklearn.model_selection import cross_val_score

from sklearn.linear_model import LogisticRegression

from  sklearn.linear_model import LinearRegression

from sklearn.naive_bayes import GaussianNB

from sklearn.model_selection import validation_curve

from sklearn.preprocessing import binarize

import matplotlib.pyplot as plot

from sklearn import linear_model

from IPython.display import Image

from sklearn.preprocessing import PolynomialFeatures

from sklearn import tree

import sklearn.preprocessing as prep

import pydotplus

import math
```

```
from sklearn.linear_model import RidgeCV
import sklearn.linear_model as linear_model
from sklearn.model_selection import GridSearchCV
from sklearn.cluster import KMeans
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

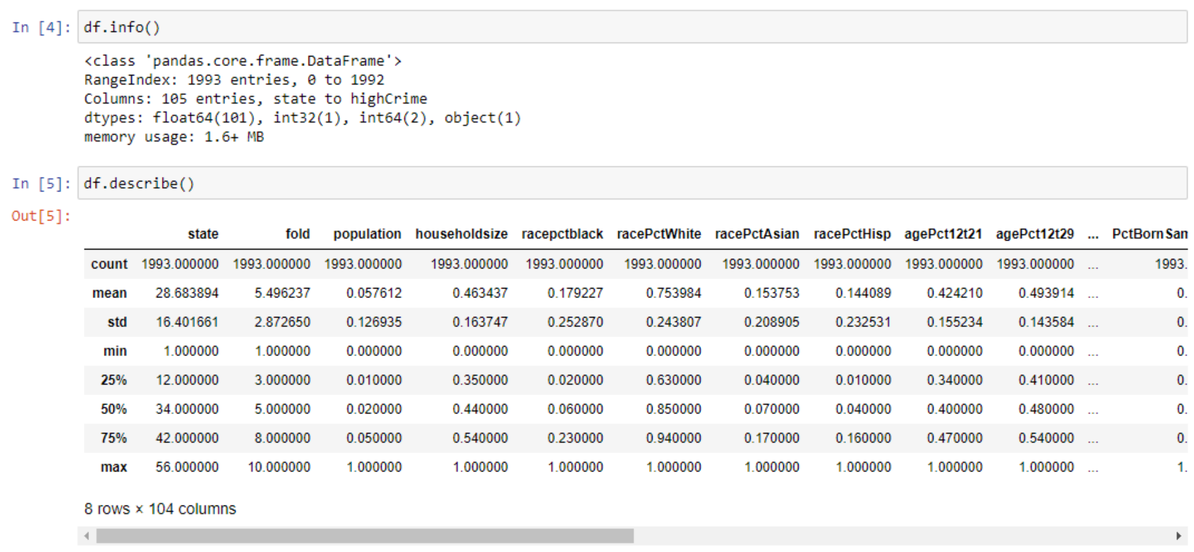
#importing the dataset (After Creating a new field “highCrime” which is true if the crime rate per capita (ViolentCrimesPerPop) is greater than 0.1, and false otherwise.)

```
df=pd.read_csv('communities-crime-clean.csv')
df['highCrime'] = np.where(df['ViolentCrimesPerPop']>0.1, 1, 0)
df
```



	state	communityname	fold	population	householdsize	racepctblack	racePctWhite	racePctAsian	racePctHisp	agePct12t21	...	PctBornSameState	PctS
0	1	Alabastercity	7	0.01	0.61	0.21	0.83	0.02	0.01	0.41	...	0.70	
1	1	AlexanderCitycity	10	0.01	0.41	0.55	0.57	0.01	0.00	0.47	...	0.93	
2	1	Annistoncity	3	0.03	0.34	0.86	0.30	0.04	0.01	0.41	...	0.77	
3	1	Athenscity	8	0.01	0.38	0.35	0.71	0.04	0.01	0.39	...	0.78	
4	1	Auburncity	1	0.04	0.37	0.32	0.70	0.21	0.02	1.00	...	0.49	
...	...	...	...	...	...	...	...	...	...	...	...	...	
1988	56	Gillettecity	9	0.01	0.53	0.00	0.96	0.02	0.06	0.47	...	0.32	
1989	56	GreenRivercity	9	0.00	0.67	0.01	0.91	0.03	0.21	0.56	...	0.35	
1990	56	Laramiecity	3	0.03	0.40	0.02	0.90	0.14	0.12	0.89	...	0.37	
1991	56	RockSpringscity	7	0.01	0.45	0.02	0.92	0.06	0.14	0.48	...	0.46	
1992	56	Sheridancity	8	0.01	0.29	0.00	0.97	0.03	0.04	0.40	...	0.45	

Performing data analysis:



```
In [4]: df.info()
```

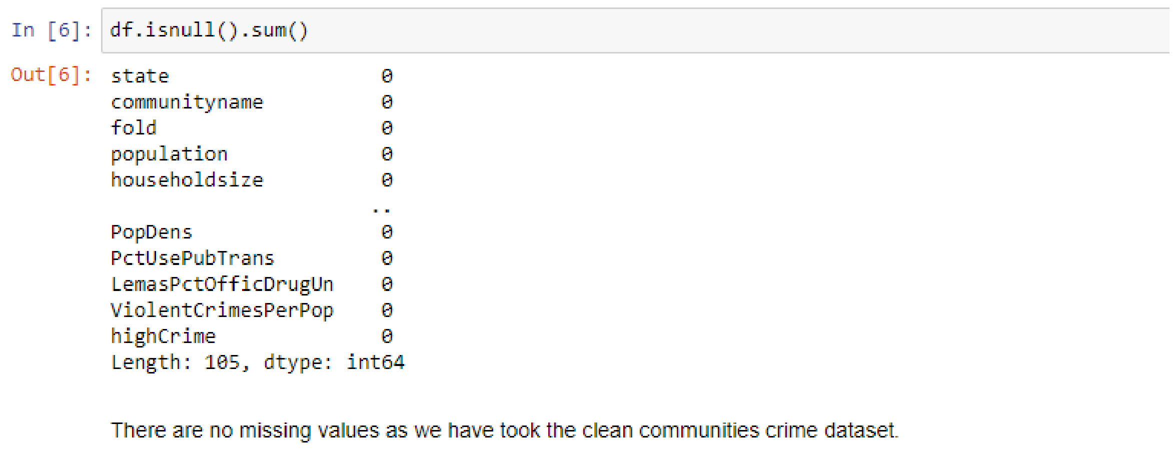
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1993 entries, 0 to 1992
Columns: 105 entries, state to highCrime
dtypes: float64(101), int32(1), int64(2), object(1)
memory usage: 1.6+ MB
```

```
In [5]: df.describe()
```

Out[5]:

	state	fold	population	householdsize	racepctblack	racePctWhite	racePctAsian	racePctHisp	agePct12t21	agePct12t29	...	PctBornSan
count	1993.000000	1993.000000	1993.000000	1993.000000	1993.000000	1993.000000	1993.000000	1993.000000	1993.000000	1993.000000	...	1993.
mean	28.683894	5.496237	0.057612	0.463437	0.179227	0.753984	0.153753	0.144089	0.424210	0.493914	...	0.
std	16.401661	2.872650	0.126935	0.163747	0.252870	0.243807	0.208905	0.232531	0.155234	0.143584	...	0.
min	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.
25%	12.000000	3.000000	0.010000	0.350000	0.020000	0.630000	0.040000	0.010000	0.340000	0.410000	...	0.
50%	34.000000	5.000000	0.020000	0.440000	0.060000	0.850000	0.070000	0.040000	0.400000	0.480000	...	0.
75%	42.000000	8.000000	0.050000	0.540000	0.230000	0.940000	0.170000	0.160000	0.470000	0.540000	...	0.
max	56.000000	10.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	1.

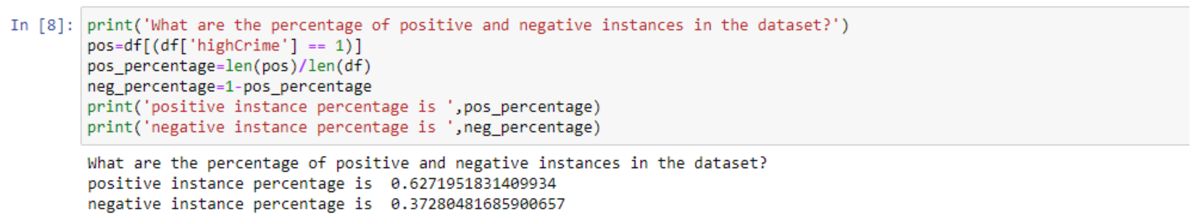
8 rows × 104 columns



```
In [6]: df.isnull().sum()
```

```
Out[6]: state                   0
        communityname           0
        fold                    0
        population              0
        householdsize           0
                               ..
        PopDens                 0
        PctUsePubTrans          0
        LemasPctOfficDrugUn     0
        ViolentCrimesPerPop     0
        highCrime               0
        Length: 105, dtype: int64
```

There are no missing values as we have took the clean communities crime dataset.



```
In [8]: print('What are the percentage of positive and negative instances in the dataset?')
        pos=df[(df['highCrime'] == 1)]
        pos_percentage=len(pos)/len(df)
        neg_percentage=1-pos_percentage
        print('positive instance percentage is ',pos_percentage)
        print('negative instance percentage is ',neg_percentage)
```

```
What are the percentage of positive and negative instances in the dataset?
positive instance percentage is  0.6271951831409934
negative instance percentage is  0.37280481685900657
```

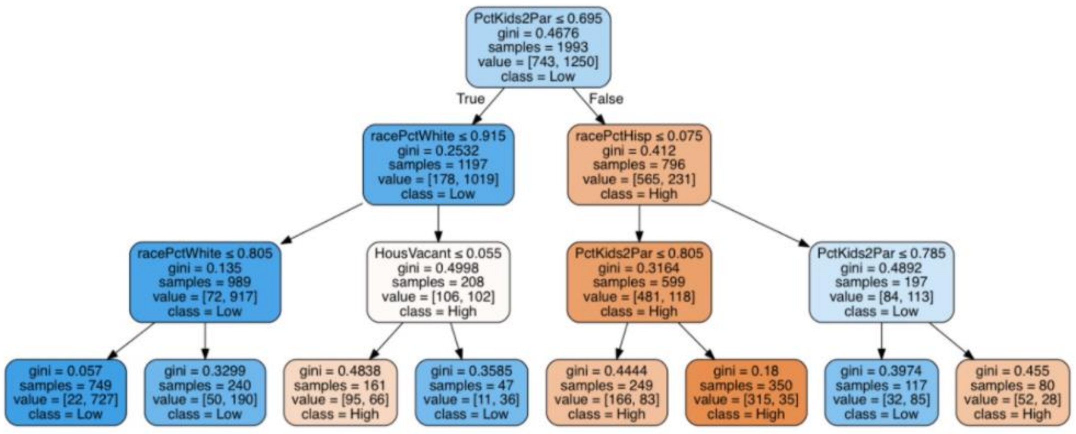
1) **DECISION TREE:**

from sklearn.model_selection import cross_val_score
from sklearn import tree
initial=pd.read_csv('communities-crime-clean.csv')
initial = initial.drop('communityname', 1)
initial = initial.drop('ViolentCrimesPerPop', 1)
initial = initial.drop('fold', 1)

initial = initial.drop('state', 1)
Y = df['highCrime']
clf = tree.DecisionTreeClassifier(max_depth=3)
#clf = tree.DecisionTreeClassifier()

```
clf = clf.fit(initial, Y)
clf
y_pred = clf.predict(initial)
list(initial)
feature_name=list(initial)
import pydotplus
from IPython.display import Image
classname=['High','Low']
dot_data = tree.export_graphviz(clf, out_file=None,
                    feature_names=list(initial),
                    class_names=classname,
                    filled=True, rounded=True,
                    special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data)
graph
```

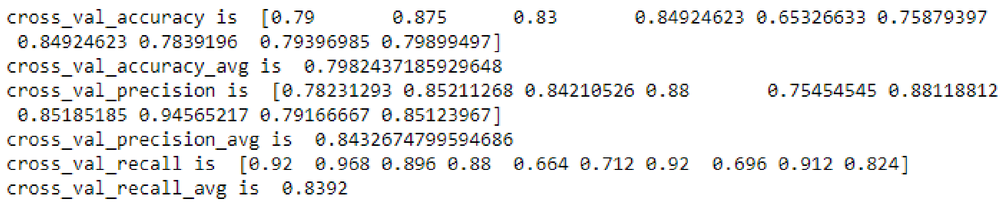
```
Out[10]: <pydotplus.graphviz.Dot at 0x1c31960bcc8>
```



Applying cross-validation (cross_val_score) to do 10-fold cross-validation to estimate the out-of-training accuracy of decision tree learning for this task.

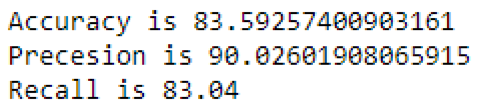
```
from sklearn.model_selection import cross_val_score
fold=df['fold']
scores = cross_val_score(clf, initial, Y,fold,'accuracy',10)
print('cross_val_accuracy is ',scores)
print('cross_val_accuracy_avg is ',np.array(scores).mean())
scores = cross_val_score(clf, initial, Y,fold,'precision',10)
print('cross_val_precision is ',scores)
print('cross_val_precision_avg is ',np.array(scores).mean())
scores = cross_val_score(clf, initial, Y,fold,'recall',10)
```

```
print('cross_val_recall is ',scores)
print('cross_val_recall_avg is ',np.array(scores).mean())
```



```
cross_val_accuracy is  [0.79       0.875      0.83       0.84924623 0.65326633 0.75879397
 0.84924623 0.7839196  0.79396985 0.79899497]
cross_val_accuracy_avg is  0.7982437185929648
cross_val_precision is  [0.78231293 0.85211268 0.84210526 0.88       0.75454545 0.88118812
 0.85185185 0.94565217 0.79166667 0.85123967]
cross_val_precision_avg is  0.8432674799594686
cross_val_recall is  [0.92  0.968 0.896 0.88  0.664 0.712 0.92  0.696 0.912 0.824]
cross_val_recall_avg is  0.8392
```

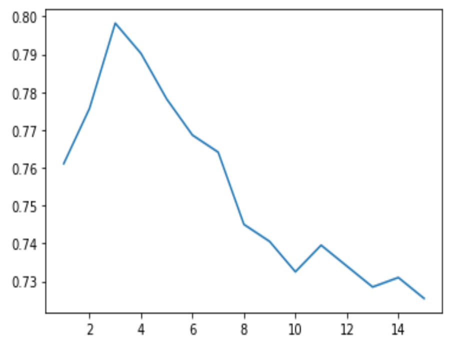
```
from sklearn.metrics import accuracy_score
Decision_tree_accuracy= accuracy_score(Y,y_pred)*100
print ('Accuracy is', accuracy_score(Y,y_pred)*100)
from sklearn.metrics import precision_score
print ('Precesion is', precision_score(Y,y_pred)*100)
from sklearn.metrics import recall_score
print ('Recall is', recall_score(Y,y_pred)*100)
```



```
Accuracy is 83.59257400903161
Precesion is 90.02601908065915
Recall is 83.04
```

Plotting a graph to prove that the max_depth can be 3.

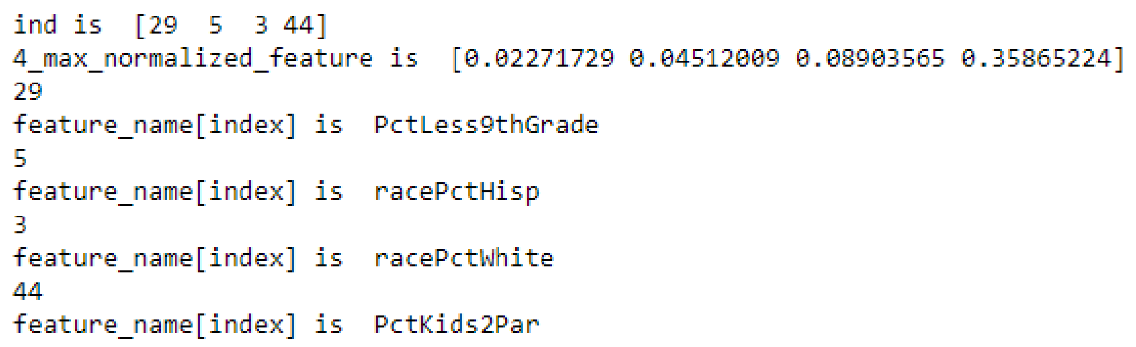
```
y=[]
x=[]
for i in range (1,16):
    clf = tree.DecisionTreeClassifier(max_depth=i)
    clf = clf.fit(initial, Y)
    y_pred = clf.predict(initial)
    scores = cross_val_score(clf, initial, Y,None,'accuracy',cv=10)
    y.append(np.array(scores).mean())
    x.append(i)


plt.plot(x, y)

plt.show()

print('',y)
```



```
[0.7611130653266331, 0.7756934673366834, 0.7982437185929648, 0.790246231155779, 0.7781859296482412, 0.7686532663316583, 0.7641
331658291457, 0.745047738693467, 0.7405326633165829, 0.7325226130653266, 0.7395502512562814, 0.7340226130653266, 0.72850753768
84421, 0.7310050251256281, 0.7254798994974875]
```

According to the plot above, we can get the best number of max_depth feeding in DecisionTreeClassifier. With the increasing number of max_depth, the mean of cross_val_score_accuracy keeps growing up and then starts to decline on number 3. Therefore, max_depth =3 in DecisionTreeClassifier can get the best performance to analyze the dataset.

```
feature_selection = clf.feature_importances_


ind = np.argpartition(feature_selection, -4)[-4:]

print('ind is ',ind)
print('4_max_normalized_feature is ',feature_selection[ind])

for x in range(0, len(ind)):
    index=ind[x]
    print(index)
    print('feature_name[index] is ',feature_name[index])
```



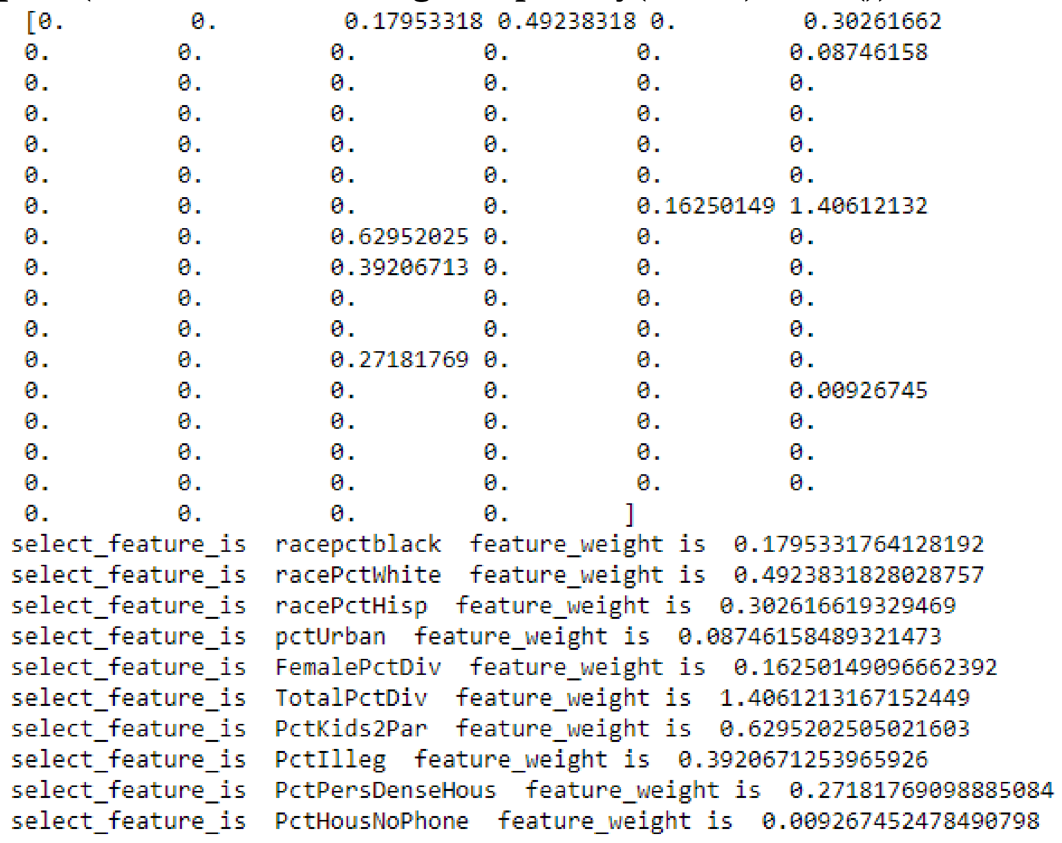
```
ind is  [29  5  3 44]
4_max_normalized_feature is  [0.02271729 0.04512009 0.08903565 0.35865224]
29
feature_name[index] is  PctLess9thGrade
5
feature_name[index] is  racePctHisp
3
feature_name[index] is  racePctWhite
44
feature_name[index] is  PctKids2Par
```

The top main feature is PctKids2Par because it is the split point of the tree. This feature depicts the kids with two parents.

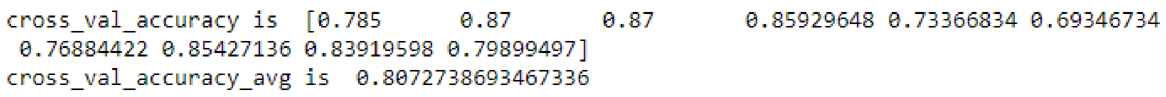
2) LINEAR SVC:

```
from sklearn import svm
lin_svc = svm.LinearSVC(C=0.01447, penalty="l1",
dual=False).fit(initial, Y)
# using L1-norm (sparsity method) to make unless feature weight
become 0 , C value increase->more complex model having more
weight
feature_weight=abs(lin_svc.coef_[0])
print("",feature_weight)
for i in range(0,len(feature_weight)):
    if(feature_weight[i]!=0):
        print('select_feature_is ',feature_name[i], ' feature_weight is ',
feature_weight[i])

from sklearn.model_selection import cross_val_score
fold=df['fold']
scores = cross_val_score(lin_svc, initial, Y,fold,'accuracy',10)
print('cross_val_accuracy is ',scores)
SVC_accuracy=np.array(scores).mean() * 100
print('cross_val_accuracy_avg is ',np.array(scores).mean())
scores = cross_val_score(lin_svc, initial, Y,fold,'precision',10)
print('cross_val_precision is ',scores)
print('cross_val_precision_avg is ',np.array(scores).mean())
scores = cross_val_score(lin_svc, initial, Y,fold,'recall',10)
```

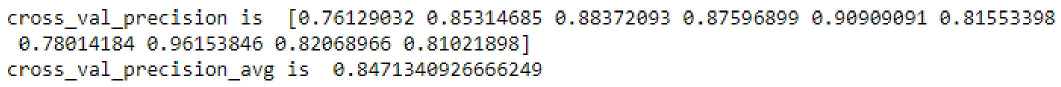
print('cross_val_recall is ',scores)
print('cross_val_recall_avg is ',np.array(scores).mean())



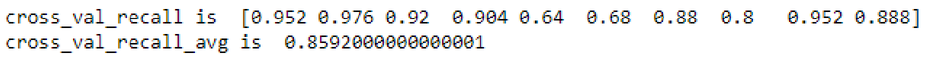
```
 [0.         0.         0.17953318 0.49238318 0.         0.30261662
 0.         0.         0.         0.         0.         0.08746158
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.16250149 1.40612132
 0.         0.         0.62952025 0.         0.         0.
 0.         0.         0.39206713 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.27181769 0.         0.         0.
 0.         0.         0.         0.         0.         0.00926745
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
select_feature_is  racepctblack  feature_weight is  0.1795331764128192
select_feature_is  racePctWhite  feature_weight is  0.4923831828028757
select_feature_is  racePctHisp  feature_weight is  0.302616619329469
select_feature_is  pctUrban  feature_weight is  0.08746158489321473
select_feature_is  FemalePctDiv  feature_weight is  0.16250149096662392
select_feature_is  TotalPctDiv  feature_weight is  1.4061213167152449
select_feature_is  PctKids2Par  feature_weight is  0.6295202505021603
select_feature_is  PctIlleg  feature_weight is  0.3920671253965926
select_feature_is  PctPersDenseHous  feature_weight is  0.27181769098885084
select_feature_is  PctHousNoPhone  feature_weight is  0.009267452478490798
```



```
cross_val_accuracy is  [0.785      0.87       0.87       0.85929648 0.73366834 0.69346734
 0.76884422 0.85427136 0.83919598 0.79899497]
cross_val_accuracy_avg is  0.8072738693467336
```



```
cross_val_precision is  [0.76129032 0.85314685 0.88372093 0.87596899 0.90909091 0.81553398
 0.78014184 0.96153846 0.82068966 0.81021898]
cross_val_precision_avg is  0.8471340926666249
```

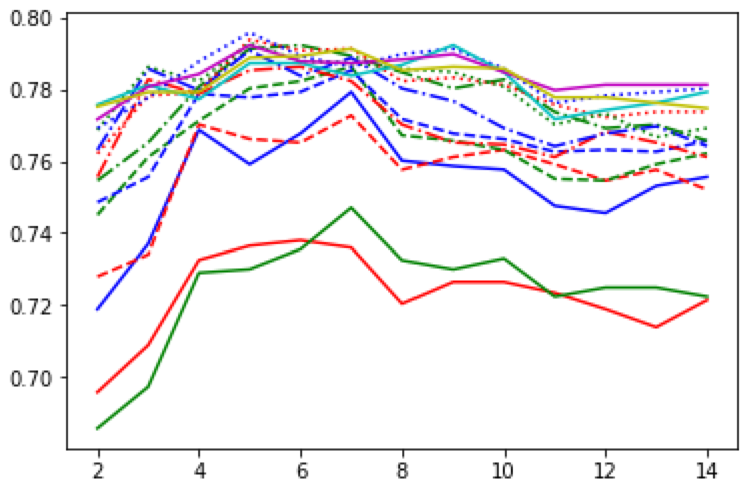


```
cross_val_recall is  [0.952 0.976 0.92  0.904 0.64  0.68  0.88  0.8   0.952 0.888]
cross_val_recall_avg is  0.8592000000000001
```

We can see that the accuracy and recall are higher in case of Decision Trees but the precision is higher in case of linear SVM. Precision would be the best criteria to compare and it is higher in case of linear SVM so the results are better in this case.
In comparison with DecisionTree, LinearSVC improves accuracy, precision and recall for this dataset. This could be explained by the fact that LinearSVC is able to find an optimal linear sepparating hyperplane for the dataset, while Decision Tree is only able to use axis alligned planes to splint in a heirarchical fashion to split the dataset. Interestingly LinearSVC has a different most predictive feature than the previous models, TotalPctDiv. PctKids2Par is the second most predictive feature for LinearSVC, so it still maintains high importance.

3) KNN

```
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier

df=pd.read_csv(r"communities-crime-clean.csv")
df['highCrime'] = np.where(df['ViolentCrimesPerPop']>0.1, 1, 0)
initial=pd.read_csv(r"communities-crime-clean.csv")
Y = df['highCrime']
fold=df['fold']
state=df['state']
community=df['communityname']
initial = initial.drop('fold', 1)
initial = initial.drop('state', 1)
initial = initial.drop('communityname', 1)
initial = initial.drop('ViolentCrimesPerPop', 1)

x=[]
y=[]
for k in range (1,16):
   y.append([])
for i in range (2,15):
   pca = PCA(n_components=i)
   pca.fit(initial)
   pcdf = pca.transform(initial)
   for j in range (1,16):
      knn = KNeighborsClassifier(j)
      knn.fit(pcdf,Y)
      scores = cross_val_score(knn,pcdf,Y,fold,'accuracy',10)
      y[j-1].append(np.mean(scores))
   x.append(i)

plt.plot(x,y[0],'r-',
       x,y[1],'g-',
       x,y[2],'b-',
       x,y[3],'r--',
       x,y[4],'g--',
       x,y[5],'b--',
       x,y[6],'r-.',
       x,y[7],'g-.',
         x,y[8],'b-.',
         x,y[9],'r:',
         x,y[10],'g:',
         x,y[11],'b:',
         x,y[12],'c-',
         x,y[13],'m-',
         x,y[14],'y-')
plt.show()
```

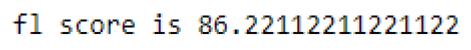


- [X-axis is number of components
- Y-axis is accuracy
    - Red-solid-line is k=1
    - Green-solid-line is k=2
    - Blue-solid-line is k=3
    - Red-dashed-line is k=4
    - Green-dashed-line is k=5
    - Blue-dashed-line is k=6
    - Red-dash-dot-line is k=7
    - Green-dash-dot-line is k=8
    - Blue-dash-dot-line is k=9
    - Red-dotted-line is k=10
    - Green-dotted-line is k=11
    - Blue-dotted-line is k=12
    - Cyan-solid-line is k=13
    - Magenta-solid-line is k=14
    - Yellow-solid-line is k=15

Blue dotted line has best performance at n-components=5 meaning k=12]

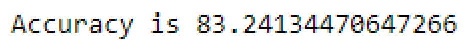
```
from sklearn.metrics import f1_score
```

```
knn = KNeighborsClassifier(n_neighbors=12)
knn.fit(pcdf,Y)
y_pred = knn.predict(pcdf)
print ('fl score is', f1_score(Y,y_pred,average="binary")*100)
```



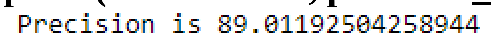
```
fl score is 86.22112211221122
```

```
print ('Accuracy is', accuracy_score(Y,y_pred)*100)
```



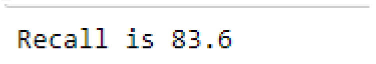
```
Accuracy is 83.24134470647266
```

```
print ('Precision is', precision_score(Y,y_pred)*100)
```



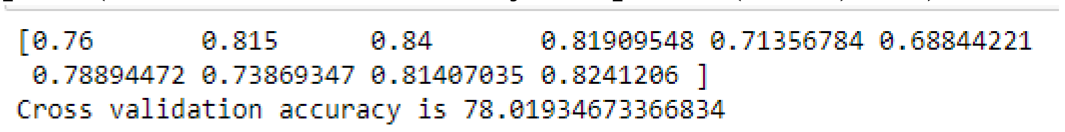
```
Precision is 89.01192504258944
```

```
print ('Recall is', recall_score(Y,y_pred)*100)
```



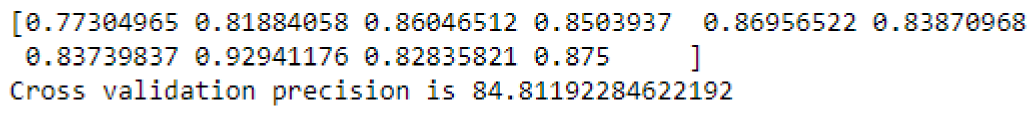
```
Recall is 83.6
```

```
scores = cross_val_score(knn,pcdf,Y,fold,'accuracy',10)
print(scores)
print ('Cross validation accuracy is', np.mean(scores)*100)
```



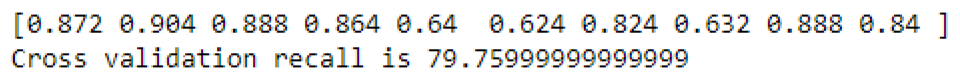
```
[0.76       0.815      0.84       0.81909548 0.71356784 0.68844221
 0.78894472 0.73869347 0.81407035 0.8241206 ]
Cross validation accuracy is 78.01934673366834
```

```
scores = cross_val_score(knn,pcdf,Y,fold,'precision',10)
print(scores)
print ('Cross validation precision is', np.mean(scores)*100)
```



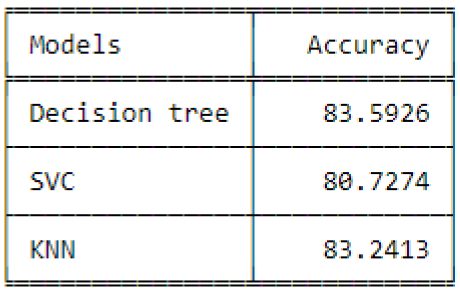
```
[0.77304965 0.81884058 0.86046512 0.8503937  0.86956522 0.83870968
 0.83739837 0.92941176 0.82835821 0.875     ]
Cross validation precision is 84.81192284622192
```

```
scores = cross_val_score(knn,pcdf,Y,fold,'recall',10)
print(scores)
print ('Cross validation recall is', np.mean(scores)*100)
```



```
[0.872 0.904 0.888 0.864 0.64  0.624 0.824 0.632 0.888 0.84 ]
Cross validation recall is 79.75999999999999
```

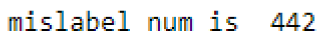
```
# Tabulating all the scores obatining using pca as well as the score of
the unreduced model
from tabulate import tabulate
info = {'Models':['Decision tree','SVC','KNN'],
'Accuracy':[Decision_tree_accuracy,SVC_accuracy,
accuracy_score(Y,y_pred)*100]}
print(tabulate(info,headers='keys',tablefmt='fancy_grid'))
```



```
+===============+============+
| Models        |   Accuracy |
+===============+============+
| Decision tree |    83.5926 |
+---------------+------------+
| SVC           |    80.7274 |
+---------------+------------+
| KNN           |    83.2413 |
+===============+============+
```

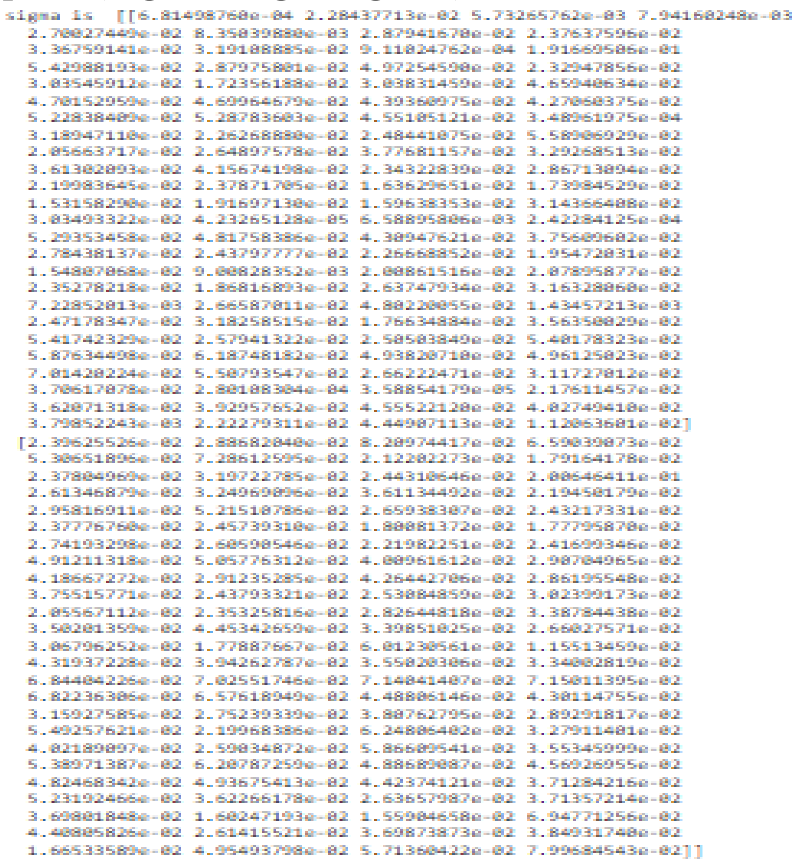
4) GAUSSIAN NAIVES BAYES:

from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
y_pred = gnb.fit(initial, Y).predict(initial)
print("mislabel num is ",(Y != y_pred).sum())



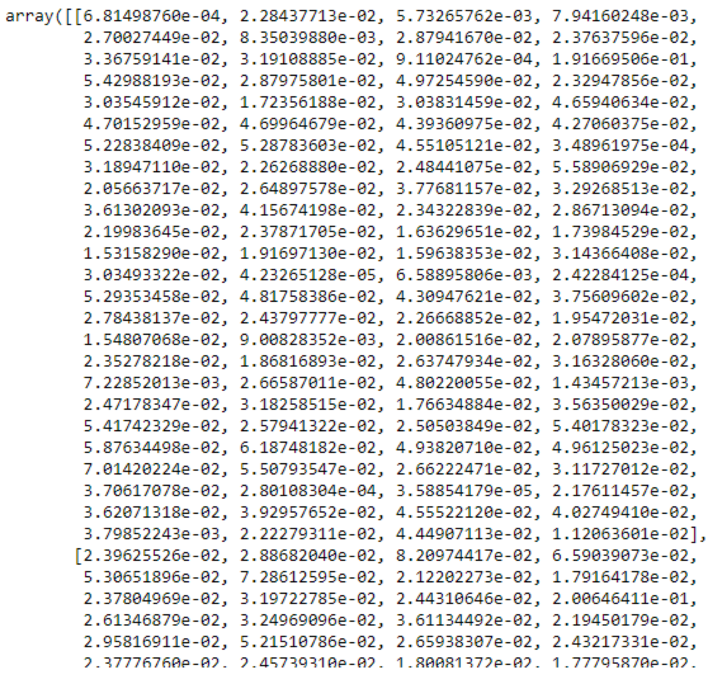
```
 mislabel num is  442
```

print ('sigma is ',gnb.sigma_)

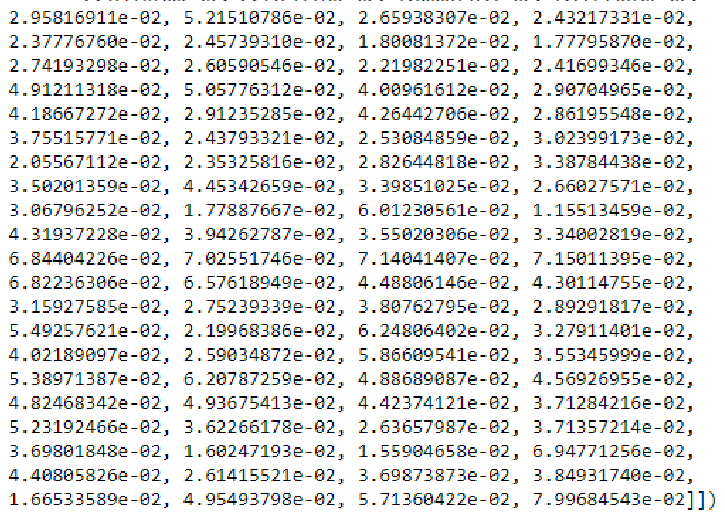


```
sigma is  [[6.81498760e-04 2.28437713e-02 5.73265762e-03 7.94168248e-03
  2.70027449e-02 8.35039880e-03 2.87941670e-02 2.37637596e-02
  3.36759141e-02 3.19108885e-02 9.11024762e-04 1.91669506e-01
  5.42988193e-02 2.87975801e-02 4.97254590e-02 2.32947856e-02
  3.03545912e-02 1.72356188e-02 3.03831459e-02 4.65940634e-02
  4.70152959e-02 4.69964679e-02 4.39360975e-02 4.27060375e-02
  5.22838409e-02 5.28783603e-02 4.55105121e-02 3.48961975e-04
  3.18947110e-02 2.26268880e-02 2.48441075e-02 5.58906929e-02
  2.05663717e-02 2.64897578e-02 3.77681157e-02 3.29268513e-02
  3.61302093e-02 4.15674198e-02 2.34322839e-02 2.86713894e-02
  2.19983645e-02 2.37871705e-02 1.63629651e-02 1.73984529e-02
  1.53158290e-02 1.91697130e-02 1.59638353e-02 3.14366408e-02
  3.03493322e-02 4.23265128e-05 6.58895806e-03 2.42284125e-04
  5.29353458e-02 4.81758386e-02 4.30947621e-02 3.75609682e-02
  2.78438137e-02 2.43797777e-02 2.26668852e-02 1.95472831e-02
  1.54807068e-02 9.00828352e-03 2.00861516e-02 2.07895877e-02
  2.35278218e-02 1.86816893e-02 2.63747934e-02 3.16328060e-02
  7.22852013e-03 2.66587011e-02 4.80220055e-02 1.43457213e-03
  2.47178347e-02 3.18258515e-02 1.76634884e-02 3.56350829e-02
  5.41742329e-02 2.57941322e-02 2.50503849e-02 5.40178323e-02
  5.87634498e-02 6.18748182e-02 4.93820710e-02 4.96125823e-02
  7.01420224e-02 5.50793547e-02 2.66222471e-02 3.11727012e-02
  3.70617078e-02 2.80108304e-04 3.58854179e-05 2.17611457e-02
  3.62071318e-02 3.92957652e-02 4.55522120e-02 4.02749410e-02
  3.79852243e-03 2.22279311e-02 4.44987113e-02 1.12063601e-02]
 [2.39625526e-02 2.88682040e-02 8.20974417e-02 6.59039873e-02
  5.30651896e-02 7.28612595e-02 2.12202273e-02 1.79164178e-02
  2.37804969e-02 3.19722785e-02 2.44310646e-02 2.00646411e-01
  2.61346879e-02 3.24969096e-02 3.61134492e-02 2.19450179e-02
  2.95816911e-02 5.21510786e-02 2.65938307e-02 2.43217331e-02
  2.37776760e-02 2.45739310e-02 1.80081372e-02 1.77795870e-02
  2.74193298e-02 2.60590546e-02 2.21982251e-02 2.41699346e-02
  4.91211318e-02 5.05776312e-02 4.00961612e-02 2.90704965e-02
  4.18667272e-02 2.91235285e-02 4.26442706e-02 2.86195548e-02
  3.75515771e-02 2.43793321e-02 2.53084859e-02 3.02399173e-02
  2.05567112e-02 2.35325816e-02 2.82644818e-02 3.38784438e-02
  3.50201359e-02 4.45342659e-02 3.39851025e-02 2.66027571e-02
  3.06796252e-02 1.77887667e-02 6.01230561e-02 1.15513459e-02
  4.31937228e-02 3.94262787e-02 3.55020306e-02 3.34002819e-02
  6.84404226e-02 7.02551746e-02 7.14041407e-02 7.15011395e-02
  6.82236306e-02 6.57618949e-02 4.48806146e-02 4.30114755e-02
  3.15927585e-02 2.75239339e-02 3.80762795e-02 2.89291817e-02
  5.49257621e-02 2.19968386e-02 6.24806402e-02 3.27911401e-02
  4.02189097e-02 2.59034872e-02 5.86609541e-02 3.55345999e-02
  5.38971387e-02 6.20787259e-02 4.88689087e-02 4.56926955e-02
  4.82468342e-02 4.93675413e-02 4.42374121e-02 3.71284216e-02
  5.23192466e-02 3.62266178e-02 2.63657987e-02 3.71357214e-02
  3.69801848e-02 1.60247193e-02 1.55984658e-02 6.94771256e-02
  4.40805826e-02 2.61415521e-02 3.69873873e-02 3.84931740e-02
  1.66533589e-02 4.95493798e-02 5.71360422e-02 7.99684543e-02]]
```

variance=gnb.sigma_
variance

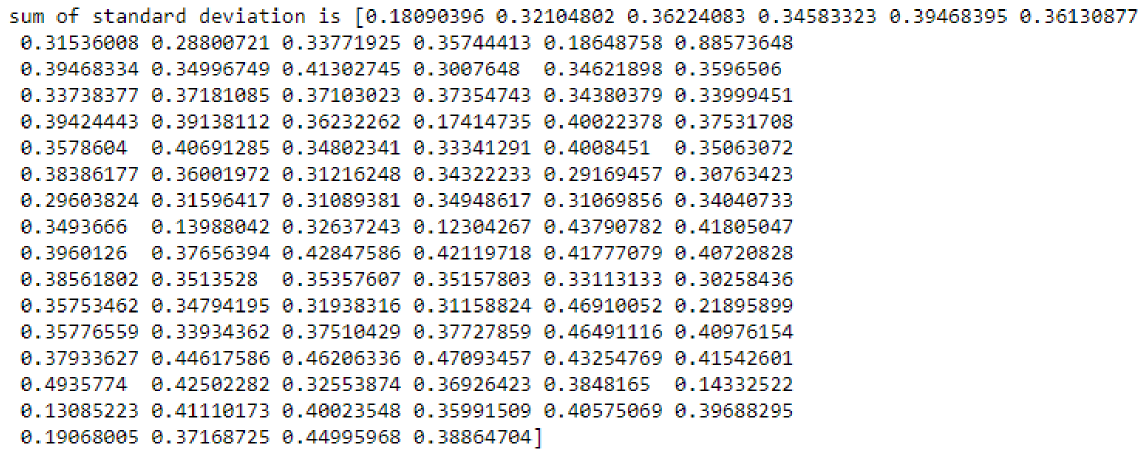


```
array([[6.81498760e-04, 2.28437713e-02, 5.73265762e-03, 7.94160248e-03,
        2.70027449e-02, 8.35039880e-03, 2.87941670e-02, 2.37637596e-02,
        3.36759141e-02, 3.19108885e-02, 9.11024762e-04, 1.91669506e-01,
        5.42988193e-02, 2.87975801e-02, 4.97254590e-02, 2.32947856e-02,
        3.03545912e-02, 1.72356188e-02, 3.03831459e-02, 4.65940634e-02,
        4.70152959e-02, 4.69964679e-02, 4.39360975e-02, 4.27060375e-02,
        5.22838409e-02, 5.28783603e-02, 4.55105121e-02, 3.48961975e-04,
        3.18947110e-02, 2.26268880e-02, 2.48441075e-02, 5.58906929e-02,
        2.05663717e-02, 2.64897578e-02, 3.77681157e-02, 3.29268513e-02,
        3.61302093e-02, 4.15674198e-02, 2.34322839e-02, 2.86713094e-02,
        2.19983645e-02, 2.37871705e-02, 1.63629651e-02, 1.73984529e-02,
        1.53158290e-02, 1.91697130e-02, 1.59638353e-02, 3.14366408e-02,
        3.03493322e-02, 4.23265128e-05, 6.58895806e-03, 2.42284125e-04,
        5.29353458e-02, 4.81758386e-02, 4.30947621e-02, 3.75609602e-02,
        2.78438137e-02, 2.43797777e-02, 2.26668852e-02, 1.95472031e-02,
        1.54807068e-02, 9.00828352e-03, 2.00861516e-02, 2.07895877e-02,
        2.35278218e-02, 1.86816893e-02, 2.63747934e-02, 3.16328060e-02,
        7.22852013e-03, 2.66587011e-02, 4.80220055e-02, 1.43457213e-03,
        2.47178347e-02, 3.18258515e-02, 1.76634884e-02, 3.56350029e-02,
        5.41742329e-02, 2.57941322e-02, 2.50503849e-02, 5.40178323e-02,
        5.87634498e-02, 6.18748182e-02, 4.93820710e-02, 4.96125023e-02,
        7.01420224e-02, 5.50793547e-02, 2.66222471e-02, 3.11727012e-02,
        3.70617078e-02, 2.80108304e-04, 3.58854179e-05, 2.17611457e-02,
        3.62071318e-02, 3.92957652e-02, 4.55522120e-02, 4.02749410e-02,
        3.79852243e-03, 2.22279311e-02, 4.44907113e-02, 1.12063601e-02],
       [2.39625526e-02, 2.88682040e-02, 8.20974417e-02, 6.59039073e-02,
        5.30651896e-02, 7.28612595e-02, 2.12202273e-02, 1.79164178e-02,
        2.37804969e-02, 3.19722785e-02, 2.44310646e-02, 2.00646411e-01,
        2.61346879e-02, 3.24969096e-02, 3.61134492e-02, 2.19450179e-02,
        2.95816911e-02, 5.21510786e-02, 2.65938307e-02, 2.43217331e-02,
        2.37776760e-02, 2.45739310e-02, 1.80081372e-02, 1.77795870e-02,
```



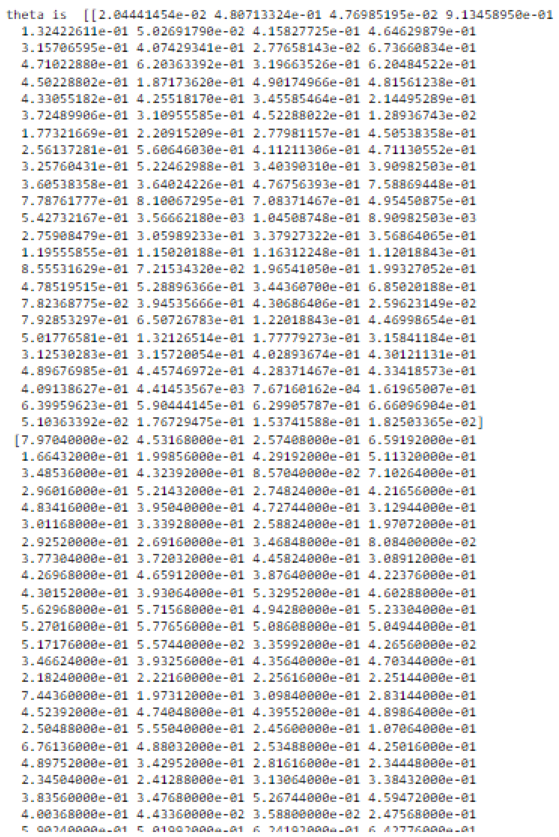
```
2.95816911e-02, 5.21510786e-02, 2.65938307e-02, 2.43217331e-02,
2.37776760e-02, 2.45739310e-02, 1.80081372e-02, 1.77795870e-02,
2.74193298e-02, 2.60590546e-02, 2.21982251e-02, 2.41699346e-02,
4.91211318e-02, 5.05776312e-02, 4.00961612e-02, 2.90704965e-02,
4.18667272e-02, 2.91235285e-02, 4.26442706e-02, 2.86195548e-02,
3.75515771e-02, 2.43793321e-02, 2.53084859e-02, 3.02399173e-02,
2.05567112e-02, 2.35325816e-02, 2.82644818e-02, 3.38784438e-02,
3.50201359e-02, 4.45342659e-02, 3.39851025e-02, 2.66027571e-02,
3.06796252e-02, 1.77887667e-02, 6.01230561e-02, 1.15513459e-02,
4.31937228e-02, 3.94262787e-02, 3.55020306e-02, 3.34002819e-02,
6.84404226e-02, 7.02551746e-02, 7.14041407e-02, 7.15011395e-02,
6.82236306e-02, 6.57618949e-02, 4.48806146e-02, 4.30114755e-02,
3.15927585e-02, 2.75239339e-02, 3.80762795e-02, 2.89291817e-02,
5.49257621e-02, 2.19968386e-02, 6.24806402e-02, 3.27911401e-02,
4.02189097e-02, 2.59034872e-02, 5.86609541e-02, 3.55345999e-02,
5.38971387e-02, 6.20787259e-02, 4.88689087e-02, 4.56926955e-02,
4.82468342e-02, 4.93675413e-02, 4.42374121e-02, 3.71284216e-02,
5.23192466e-02, 3.62266178e-02, 2.63657987e-02, 3.71357214e-02,
3.69801848e-02, 1.60247193e-02, 1.55904658e-02, 6.94771256e-02,
4.40805826e-02, 2.61415521e-02, 3.69873873e-02, 3.84931740e-02,
1.66533589e-02, 4.95493798e-02, 5.71360422e-02, 7.99684543e-02]])
```

standard_deviation =np.sqrt(variance)
sum_standard=standard_deviation[0]+standard_deviation[1]
print('sum of standard deviation is',sum_standard);



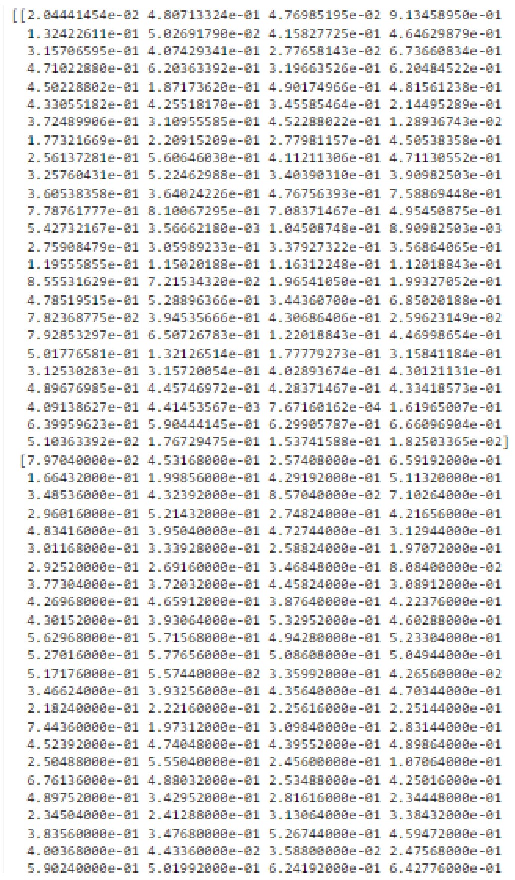
```
sum of standard deviation is [0.18090396 0.32104802 0.36224083 0.34583323 0.39468395 0.36130877
 0.31536008 0.28800721 0.33771925 0.35744413 0.18648758 0.88573648
 0.39468334 0.34996749 0.41302745 0.3007648  0.34621898 0.3596506
 0.33738377 0.37181085 0.37103023 0.37354743 0.34380379 0.33999451
 0.39424443 0.39138112 0.36232262 0.17414735 0.40022378 0.37531708
 0.3578604  0.40691285 0.34802341 0.33341291 0.4008451  0.35063072
 0.38386177 0.36001972 0.31216248 0.34322233 0.29169457 0.30763423
 0.29603824 0.31596417 0.31089381 0.34948617 0.31069856 0.34040733
 0.3493666  0.13988042 0.32637243 0.12304267 0.43790782 0.41805047
 0.3960126  0.37656394 0.42847586 0.42119718 0.41777079 0.40720828
 0.38561802 0.3513528  0.35357607 0.35157803 0.33113133 0.30258436
 0.35753462 0.34794195 0.31938316 0.31158824 0.46910052 0.21895899
 0.35776559 0.33934362 0.37510429 0.37727859 0.46491116 0.40976154
 0.37933627 0.44617586 0.46206336 0.47093457 0.43254769 0.41542601
 0.4935774  0.42502282 0.32553874 0.36926423 0.3848165  0.14332522
 0.13085223 0.41110173 0.40023548 0.35991509 0.40575069 0.39688295
 0.19068005 0.37168725 0.44995968 0.38864704]
```

print ('theta is ',gnb.theta_)



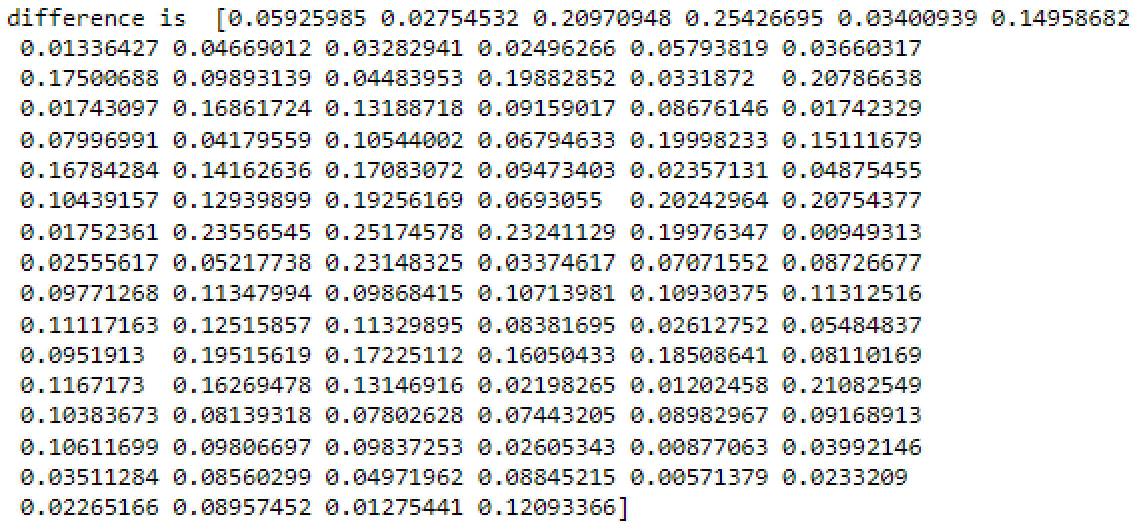
```
theta is  [[2.04441454e-02 4.80713324e-01 4.76985195e-02 9.13458950e-01
  1.32422611e-01 5.02691790e-02 4.15827725e-01 4.64629879e-01
  3.15706595e-01 4.07429341e-01 2.77658143e-02 6.73660834e-01
  4.71022880e-01 6.20363392e-01 3.19663526e-01 6.20484522e-01
  4.50228802e-01 1.87173620e-01 4.90174966e-01 4.81561238e-01
  4.33055182e-01 4.25518170e-01 3.45585464e-01 2.14495289e-01
  3.72489906e-01 3.10955585e-01 4.52288022e-01 1.28936743e-02
  1.77321669e-01 2.20915209e-01 2.77981157e-01 4.50538358e-01
  2.56137281e-01 5.60646030e-01 4.11211306e-01 4.71130552e-01
  3.25760431e-01 5.22462988e-01 3.40390310e-01 3.90982503e-01
  3.60538358e-01 3.64024226e-01 4.76756393e-01 7.58869448e-01
  7.78761777e-01 8.10067295e-01 7.08371467e-01 4.95450875e-01
  5.42732167e-01 3.56662180e-03 1.04508748e-01 8.90982503e-03
  2.75908479e-01 3.05989233e-01 3.37927322e-01 3.56864065e-01
  1.19555855e-01 1.15020188e-01 1.16312248e-01 1.12018843e-01
  8.55531629e-01 7.21534320e-02 1.96541050e-01 1.99327052e-01
  4.78519515e-01 5.28896366e-01 3.44360700e-01 6.85020188e-01
  7.82368775e-02 3.94535666e-01 4.30686406e-01 2.59623149e-02
  7.92853297e-01 6.50726783e-01 1.22018843e-01 4.46998654e-01
  5.01776581e-01 1.32126514e-01 1.77779273e-01 3.15841184e-01
  3.12530283e-01 3.15720054e-01 4.02893674e-01 4.30121131e-01
  4.89676985e-01 4.45746972e-01 4.28371467e-01 4.33418573e-01
  4.09138627e-01 4.41453567e-03 7.67160162e-04 1.61965007e-01
  6.39959623e-01 5.90444145e-01 6.29905787e-01 6.66096904e-01
  5.10363392e-02 1.76729475e-01 1.53741588e-01 1.82503365e-02]
 [7.97040000e-02 4.53168000e-01 2.57408000e-01 6.59192000e-01
  1.66432000e-01 1.99856000e-01 4.29192000e-01 5.11320000e-01
  3.48536000e-01 4.32392000e-01 8.57040000e-02 7.10264000e-01
  2.96016000e-01 5.21432000e-01 2.74824000e-01 4.21656000e-01
  4.83416000e-01 3.95040000e-01 4.72744000e-01 3.12944000e-01
  3.01168000e-01 3.33928000e-01 2.58824000e-01 1.97072000e-01
  2.92520000e-01 2.69160000e-01 3.46848000e-01 8.08400000e-02
  3.77304000e-01 3.72032000e-01 4.45824000e-01 3.08912000e-01
  4.26968000e-01 4.65912000e-01 3.87640000e-01 4.22376000e-01
  4.30152000e-01 3.93064000e-01 5.32952000e-01 4.60288000e-01
  5.62968000e-01 5.71568000e-01 4.94280000e-01 5.23304000e-01
  5.27016000e-01 5.77656000e-01 5.08608000e-01 5.04944000e-01
  5.17176000e-01 5.57440000e-02 3.35992000e-01 4.26560000e-02
  3.46624000e-01 3.93256000e-01 4.35640000e-01 4.70344000e-01
  2.18240000e-01 2.22160000e-01 2.25616000e-01 2.25144000e-01
  7.44360000e-01 1.97312000e-01 3.09840000e-01 2.83144000e-01
  4.52392000e-01 4.74048000e-01 4.39552000e-01 4.89864000e-01
  2.50488000e-01 5.55040000e-01 2.45600000e-01 1.07064000e-01
  6.76136000e-01 4.88032000e-01 2.53488000e-01 4.25016000e-01
  4.89752000e-01 3.42952000e-01 2.81616000e-01 2.34448000e-01
  2.34504000e-01 2.41288000e-01 3.13064000e-01 3.38432000e-01
  3.83560000e-01 3.47680000e-01 5.26744000e-01 4.59472000e-01
  4.00368000e-01 4.43360000e-02 3.58800000e-02 2.47568000e-01
  5.90240000e-01 5.01992000e-01 6.24192000e-01 6.42776000e-01
```

mean=gnb.theta_
print(mean)
difference=mean[0]-mean[1]



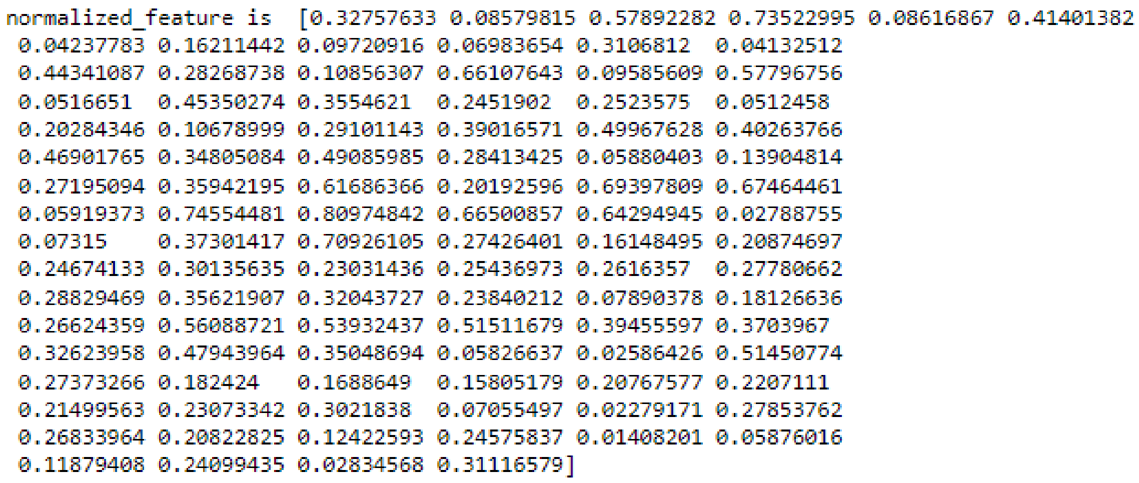
```
[[2.04441454e-02 4.80713324e-01 4.76985195e-02 9.13458950e-01
  1.32422611e-01 5.02691790e-02 4.15827725e-01 4.64629879e-01
  3.15706595e-01 4.07429341e-01 2.77658143e-02 6.73660834e-01
  4.71022880e-01 6.20363392e-01 3.19663526e-01 6.20484522e-01
  4.50228802e-01 1.87173620e-01 4.90174966e-01 4.81561238e-01
  4.33055182e-01 4.25518170e-01 3.45585464e-01 2.14495289e-01
  3.72489906e-01 3.10955585e-01 4.52288022e-01 1.28936743e-02
  1.77321669e-01 2.20915209e-01 2.77981157e-01 4.50538358e-01
  2.56137281e-01 5.60646030e-01 4.11211306e-01 4.71130552e-01
  3.25760431e-01 5.22462988e-01 3.40390310e-01 3.90982503e-01
  3.60538358e-01 3.64024226e-01 4.76756393e-01 7.58869448e-01
  7.78761777e-01 8.10067295e-01 7.08371467e-01 4.95450875e-01
  5.42732167e-01 3.56662180e-03 1.04508748e-01 8.90982503e-03
  2.75908479e-01 3.05989233e-01 3.37927322e-01 3.56864065e-01
  1.19555855e-01 1.15020188e-01 1.16312248e-01 1.12018843e-01
  8.55531629e-01 7.21534320e-02 1.96541050e-01 1.99327052e-01
  4.78519515e-01 5.28896366e-01 3.44360700e-01 6.85020188e-01
  7.82368775e-02 3.94535666e-01 4.30686406e-01 2.59623149e-02
  7.92853297e-01 6.50726783e-01 1.22018843e-01 4.46998654e-01
  5.01776581e-01 1.32126514e-01 1.77779273e-01 3.15841184e-01
  3.12530283e-01 3.15720054e-01 4.02893674e-01 4.30121131e-01
  4.89676985e-01 4.45746972e-01 4.28371467e-01 4.33418573e-01
  4.09138627e-01 4.41453567e-03 7.67160162e-04 1.61965007e-01
  6.39959623e-01 5.90444145e-01 6.29905787e-01 6.66096904e-01
  5.10363392e-02 1.76729475e-01 1.53741588e-01 1.82503365e-02]
 [7.97040000e-02 4.53168000e-01 2.57408000e-01 6.59192000e-01
  1.66432000e-01 1.99856000e-01 4.29192000e-01 5.11320000e-01
  3.48536000e-01 4.32392000e-01 8.57040000e-02 7.10264000e-01
  2.96016000e-01 5.21432000e-01 2.74824000e-01 4.21656000e-01
  4.83416000e-01 3.95040000e-01 4.72744000e-01 3.12944000e-01
  3.01168000e-01 3.33928000e-01 2.58824000e-01 1.97072000e-01
  2.92520000e-01 2.69160000e-01 3.46848000e-01 8.08400000e-02
  3.77304000e-01 3.72032000e-01 4.45824000e-01 3.08912000e-01
  4.26968000e-01 4.65912000e-01 3.87640000e-01 4.22376000e-01
  4.30152000e-01 3.93064000e-01 5.32952000e-01 4.60288000e-01
  5.62968000e-01 5.71568000e-01 4.94280000e-01 5.23304000e-01
  5.27016000e-01 5.77656000e-01 5.08608000e-01 5.04944000e-01
  5.17176000e-01 5.57440000e-02 3.35992000e-01 4.26560000e-02
  3.46624000e-01 3.93256000e-01 4.35640000e-01 4.70344000e-01
  2.18240000e-01 2.22160000e-01 2.25616000e-01 2.25144000e-01
  7.44360000e-01 1.97312000e-01 3.09840000e-01 2.83144000e-01
  4.52392000e-01 4.74048000e-01 4.39552000e-01 4.89864000e-01
  2.50488000e-01 5.55040000e-01 2.45600000e-01 1.07064000e-01
  6.76136000e-01 4.88032000e-01 2.53488000e-01 4.25016000e-01
  4.89752000e-01 3.42952000e-01 2.81616000e-01 2.34448000e-01
  2.34504000e-01 2.41288000e-01 3.13064000e-01 3.38432000e-01
  3.83560000e-01 3.47680000e-01 5.26744000e-01 4.59472000e-01
  4.00368000e-01 4.43360000e-02 3.58800000e-02 2.47568000e-01
  5.90240000e-01 5.01992000e-01 6.24192000e-01 6.42776000e-01
```

```
print('difference is ',abs(difference))
```



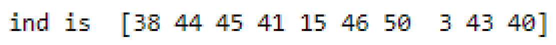
```
difference is  [0.05925985 0.02754532 0.20970948 0.25426695 0.03400939 0.14958682
 0.01336427 0.04669012 0.03282941 0.02496266 0.05793819 0.03660317
 0.17500688 0.09893139 0.04483953 0.19882852 0.0331872  0.20786638
 0.01743097 0.16861724 0.13188718 0.09159017 0.08676146 0.01742329
 0.07996991 0.04179559 0.10544002 0.06794633 0.19998233 0.15111679
 0.16784284 0.14162636 0.17083072 0.09473403 0.02357131 0.04875455
 0.10439157 0.12939899 0.19256169 0.0693055  0.20242964 0.20754377
 0.01752361 0.23556545 0.25174578 0.23241129 0.19976347 0.00949313
 0.02555617 0.05217738 0.23148325 0.03374617 0.07071552 0.08726677
 0.09771268 0.11347994 0.09868415 0.10713981 0.10930375 0.11312516
 0.11117163 0.12515857 0.11329895 0.08381695 0.02612752 0.05484837
 0.0951913  0.19515619 0.17225112 0.16050433 0.18508641 0.08110169
 0.1167173  0.16269478 0.13146916 0.02198265 0.01202458 0.21082549
 0.10383673 0.08139318 0.07802628 0.07443205 0.08982967 0.09168913
 0.10611699 0.09806697 0.09837253 0.02605343 0.00877063 0.03992146
 0.03511284 0.08560299 0.04971962 0.08845215 0.00571379 0.0233209
 0.02265166 0.08957452 0.01275441 0.12093366]
```

```
normalized_feature=abs(difference)/sum_standard
print('normalized_feature is ',normalized_feature)
```



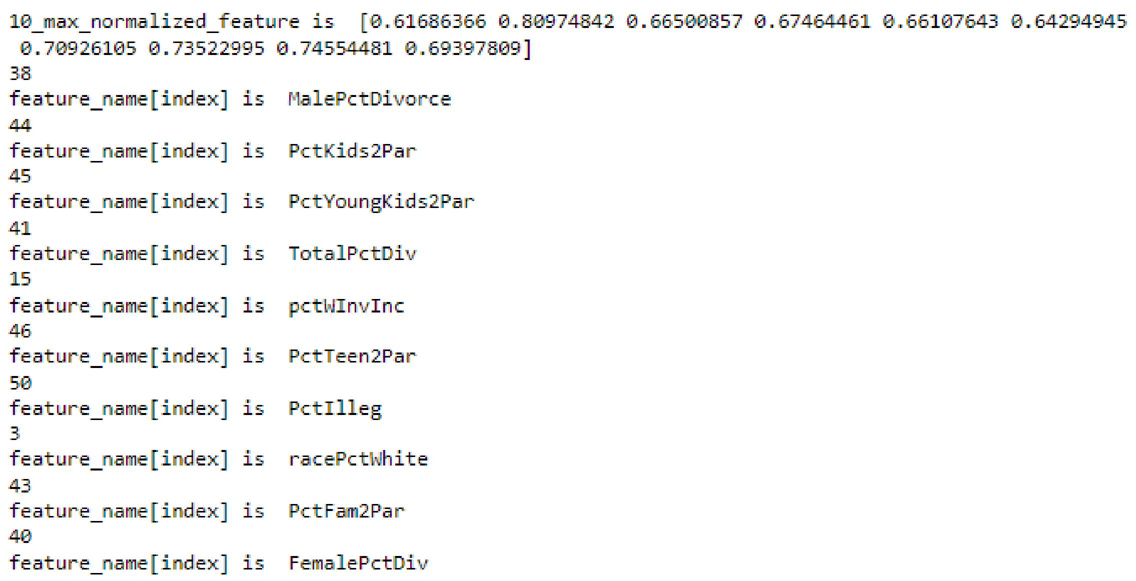
```
normalized_feature is  [0.32757633 0.08579815 0.57892282 0.73522995 0.08616867 0.41401382
 0.04237783 0.16211442 0.09720916 0.06983654 0.3106812  0.04132512
 0.44341087 0.28268738 0.10856307 0.66107643 0.09585609 0.57796756
 0.0516651  0.45350274 0.3554621  0.2451902  0.2523575  0.0512458
 0.20284346 0.10678999 0.29101143 0.39016571 0.49967628 0.40263766
 0.46901765 0.34805084 0.49085985 0.28413425 0.05880403 0.13904814
 0.27195094 0.35942195 0.61686366 0.20192596 0.69397809 0.67464461
 0.05919373 0.74554481 0.80974842 0.66500857 0.64294945 0.02788755
 0.07315    0.37301417 0.70926105 0.27426401 0.16148495 0.20874697
 0.24674133 0.30135635 0.23031436 0.25436973 0.2616357  0.27780662
 0.28829469 0.35621907 0.32043727 0.23840212 0.07890378 0.18126636
 0.26624359 0.56088721 0.53932437 0.51511679 0.39455597 0.3703967
 0.32623958 0.47943964 0.35048694 0.05826637 0.02586426 0.51450774
 0.27373266 0.182424   0.1688649  0.15805179 0.20767577 0.2207111
 0.21499563 0.23073342 0.3021838  0.07055497 0.02279171 0.27853762
 0.26833964 0.20822825 0.12422593 0.24575837 0.01408201 0.05876016
 0.11879408 0.24099435 0.02834568 0.31116579]
```

```
ind = np.argpartition(normalized_feature, -10)[-10:]

print('ind is ',ind)
```



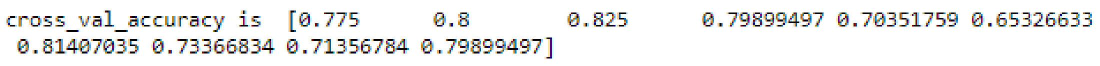
```
ind is  [38 44 45 41 15 46 50  3 43 40]
```

```
print('10_max_normalized_feature is ',normalized_feature[ind])

for x in range(0, len(ind)):
   index=ind[x]
   print(index)
   print('feature_name[index] is ',feature_name[index])
```



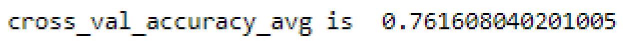
```
10_max_normalized_feature is  [0.61686366 0.80974842 0.66500857 0.67464461 0.66107643 0.64294945
 0.70926105 0.73522995 0.74554481 0.69397809]
38
feature_name[index] is  MalePctDivorce
44
feature_name[index] is  PctKids2Par
45
feature_name[index] is  PctYoungKids2Par
41
feature_name[index] is  TotalPctDiv
15
feature_name[index] is  pctWInvInc
46
feature_name[index] is  PctTeen2Par
50
feature_name[index] is  PctIlleg
3
feature_name[index] is  racePctWhite
43
feature_name[index] is  PctFam2Par
40
feature_name[index] is  FemalePctDiv
```

```
from sklearn.model_selection import cross_val_score
fold=df['fold']
scores = cross_val_score(gnb, initial, Y,fold,'accuracy',10)
print('cross_val_accuracy is ',scores)
```



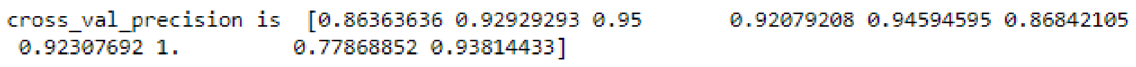
```
cross_val_accuracy is  [0.775      0.8        0.825      0.79899497 0.70351759 0.65326633
 0.81407035 0.73366834 0.71356784 0.79899497]
```

```
print('cross_val_accuracy_avg is ',np.array(scores).mean())
```



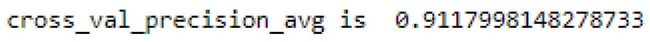
```
cross_val_accuracy_avg is  0.761608040201005
```

```
scores = cross_val_score(gnb, initial, Y,fold,'precision',10)
print('cross_val_precision is ',scores)
```



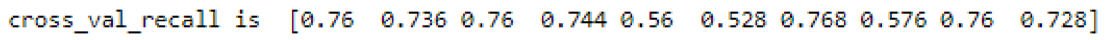
```
cross_val_precision is  [0.86363636 0.92929293 0.95       0.92079208 0.94594595 0.86842105
 0.92307692 1.         0.77868852 0.93814433]
```

```
print('cross_val_precision_avg is ',np.array(scores).mean())
```



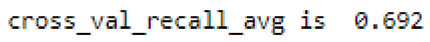
```
cross_val_precision_avg is  0.9117998148278733
```

```
scores = cross_val_score(gnb, initial, Y,fold,'recall',10)
print('cross_val_recall is ',scores)
```



```
cross_val_recall is  [0.76  0.736 0.76  0.744 0.56  0.528 0.768 0.576 0.76  0.728]
```

```
print('cross_val_recall_avg is ',np.array(scores).mean())
```

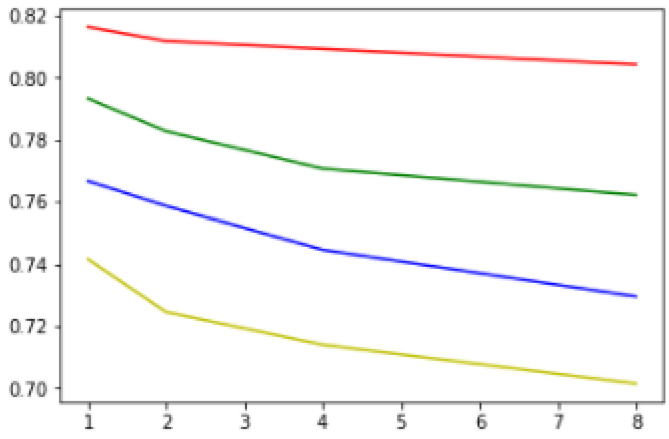


```
cross_val_recall_avg is  0.692
```

Naive Bayes on average has worse accuracy and recall than Decision Tree on the dataset. This is likely due to correlations between features in the dataset that by definition the Naive Bayes classifier assumes are conditionally independent. Interestingly, both find PctKids2Par to be the most predictive feature.

5) POLYNOMIAL SVC:

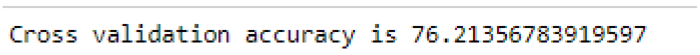
```
from sklearn.svm import SVC
x=[]
y=[]
for k in range (1,5):
    y.append([])
for i in range (0,4):
    for j in range (1,5):
        poly_svc = SVC(C=2**i, kernel='poly', degree=j).fit(initial, Y)
        scores = cross_val_score(poly_svc,initial,Y,fold,'accuracy',10)
        y[j-1].append(np.mean(scores))
    x.append(2**i)

plt.plot(x,y[0],'r-',
         x,y[1],'g-',
         x,y[2],'b-',
         x,y[3],'y-')
plt.show()
```



X-axis is error penalty (C) term Y-axis is accuracy Red=Degree 1
Green=Degree 2
Blue=Degree 3
Yellow=Degree 4

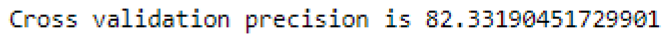
Degree 2 is picked for best performance against linear model. Error penalty value of 8 is picked for best performance.

```
poly_svc = SVC(C=8, kernel='poly', degree=2).fit(initial, Y)
scores = cross_val_score(poly_svc,initial,Y,fold,'accuracy',10)
print ('Cross validation accuracy is', np.mean(scores)*100)
```



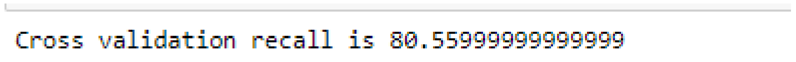
```
Cross validation accuracy is 76.21356783919597
```

```
scores = cross_val_score(poly_svc,initial,Y,fold,'precision',10)
print ('Cross validation precision is', np.mean(scores)*100)
```



```
Cross validation precision is 82.33190451729901
```

```
scores = cross_val_score(poly_svc,initial,Y,fold,'recall',10)
print ('Cross validation recall is', np.mean(scores)*100)
```



```
Cross validation recall is 80.55999999999999
```

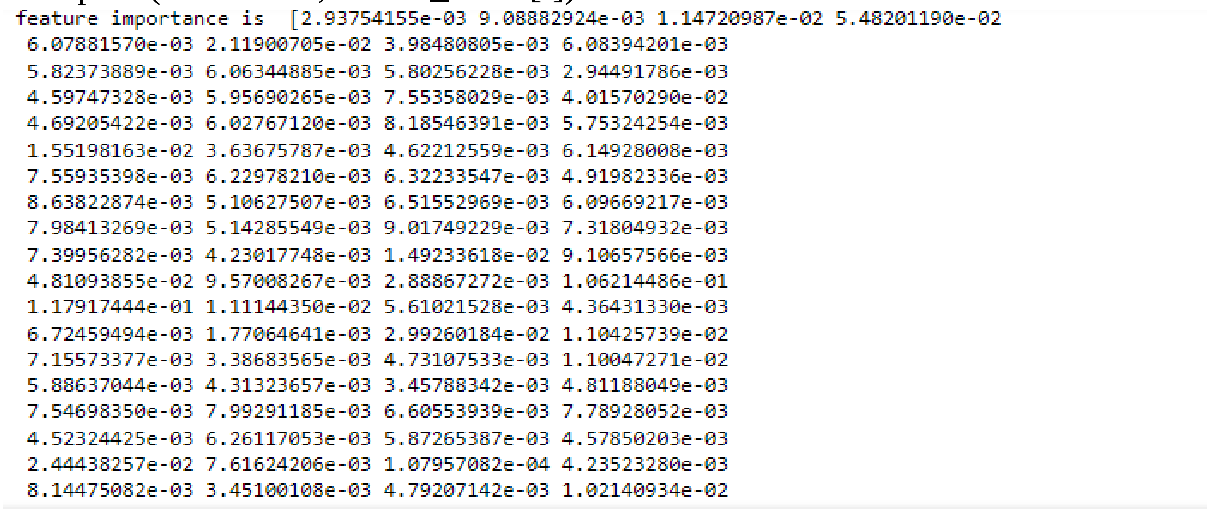
There is no reasonable method for evaluation of most predictive features with a Polynomial SVC.

Compared with Linear SVC on this data set a polynomial SVC of degree 2 is able to slightly outperform the linear SVC on classification.

6) RANDOM FOREST:

#Decision Forest using a Random Forest algorithm. Nodes are chosen from a the best split of a random subset of the features and the best of a set of random thresholds for those features.

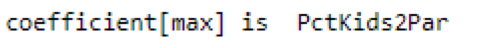
```
y_pred = clf.predict(initial)
feature_importance=clf.feature_importances_
print("feature importance is ",feature_importance)
print ("length feature array",len(feature_importance))
for i in range (0,len(feature_importance)):
    if(feature_importance[i]!=0):
        print("index is ",feature_name[i])
```



```
feature importance is  [2.93754155e-03 9.08882924e-03 1.14720987e-02 5.48201190e-02
 6.07881570e-03 2.11900705e-02 3.98480805e-03 6.08394201e-03
 5.82373889e-03 6.06344885e-03 5.80256228e-03 2.94491786e-03
 4.59747328e-03 5.95690265e-03 7.55358029e-03 4.01570290e-02
 4.69205422e-03 6.02767120e-03 8.18546391e-03 5.75324254e-03
 1.55198163e-02 3.63675787e-03 4.62212559e-03 6.14928008e-03
 7.55935398e-03 6.22978210e-03 6.32233547e-03 4.91982336e-03
 8.63822874e-03 5.10627507e-03 6.51552969e-03 6.09669217e-03
 7.98413269e-03 5.14285549e-03 9.01749229e-03 7.31804932e-03
 7.39956282e-03 4.23017748e-03 1.49233618e-02 9.10657566e-03
 4.81093855e-02 9.57008267e-03 2.88867272e-03 1.06214486e-01
 1.17917444e-01 1.11144350e-02 5.61021528e-03 4.36431330e-03
 6.72459494e-03 1.77064641e-03 2.99260184e-02 1.10425739e-02
 7.15573377e-03 3.38683565e-03 4.73107533e-03 1.10047271e-02
 5.88637044e-03 4.31323657e-03 3.45788342e-03 4.81188049e-03
 7.54698350e-03 7.99291185e-03 6.60553939e-03 7.78928052e-03
 4.52324425e-03 6.26117053e-03 5.87265387e-03 4.57850203e-03
 2.44438257e-02 7.61624206e-03 1.07957082e-04 4.23523280e-03
 8.14475082e-03 3.45100108e-03 4.79207142e-03 1.02140934e-02
```

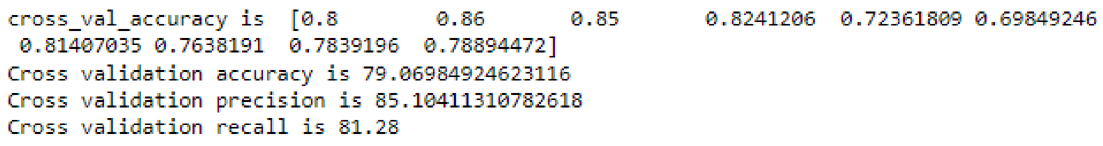
```
index_max = np.argmax(feature_importance)
print(index_max)
44
```

```
print('coefficient[max] is ',feature_name[index_max])
```



```
coefficient[max] is  PctKids2Par
```

```
fold=df['fold']
scores = cross_val_score(clf, initial, Y,fold,'accuracy',10)
print('cross_val_accuracy is ',scores)

print ('Cross validation accuracy is', np.mean(scores)*100)
scores = cross_val_score(clf,initial,Y,fold,'precision',10)
print ('Cross validation precision is', np.mean(scores)*100)
scores = cross_val_score(clf,initial,Y,fold,'recall',10)
print ('Cross validation recall is', np.mean(scores)*100)
```

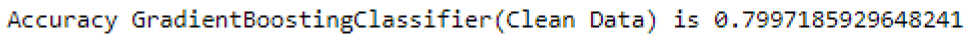


```
cross_val_accuracy is  [0.8        0.86       0.85       0.8241206  0.72361809 0.69849246
 0.81407035 0.7638191  0.7839196  0.78894472]
Cross validation accuracy is 79.06984924623116
Cross validation precision is 85.10411310782618
Cross validation recall is 81.28
```

Performance is relatively consistent with that of the standard Decision Tree. The most predictive feature is the same as Decision Tree, PctKids2Par. Random Decision Forest does not seem to offer much for the increase in complexity of the model.Random Classifier shows good results in terms of metrics and would be a good model to predict the crime data.

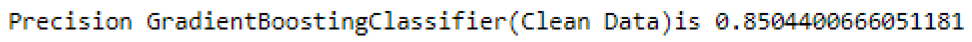
7) GRADIENT BOOSTING CLASSFIER:

```
X = df.drop('ViolentCrimesPerPop', axis=1).drop('state',
axis=1).drop('communityname', axis=1).drop('fold',
axis=1).drop('highCrime', axis=1)
features = list(X.columns)
y =df["highCrime"]
clf = GradientBoostingClassifier()
clf.fit(X, y)
acc_scores = cross_val_score(clf, X, y, cv=10,
scoring='accuracy').mean()
pre_scores = cross_val_score(clf, X, y, cv=10,
scoring='precision').mean()
rec_scores = cross_val_score(clf, X, y, cv=10, scoring='recall').mean()
print ('Accuracy GradientBoostingClassifier(Clean Data) is', acc_scores)
```



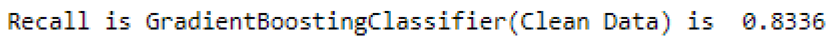
```
Accuracy GradientBoostingClassifier(Clean Data) is 0.7997185929648241
```

```
print ('Precision GradientBoostingClassifier(Clean Data)is', pre_scores)
```



```
Precision GradientBoostingClassifier(Clean Data)is 0.8504400666051181
```

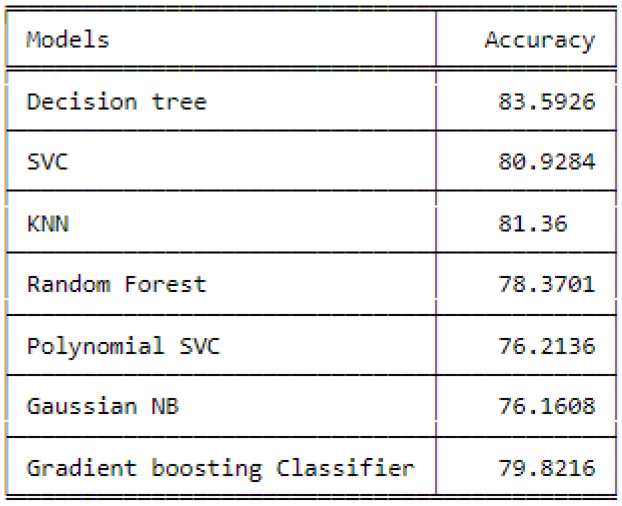
```
print ('Recall is GradientBoostingClassifier(Clean Data) is ', rec_scores)
```



```
Recall is GradientBoostingClassifier(Clean Data) is  0.8336
```

ACCURACY RESULTS:

```
# Tabulating all the scores
from tabulate import tabulate
info = {'Models':['Decision tree','SVC','KNN','Random Forest','Polynomial
SVC','Gaussian NB','Gradient boosting Classifier'],
'Accuracy':[Decision_tree_accuracy,SVC_accuracy, accuracy_score(Y,y_pred)*100,
randomforest_accuracy, poly_svc_accuracy, gaussian_accuracy, acc_scores*100]}

print(tabulate(info,headers='keys',tablefmt='fancy_grid'))
```



Models	Accuracy
Decision tree	83.5926
SVC	80.9284
KNN	81.36
Random Forest	78.3701
Polynomial SVC	76.2136
Gaussian NB	76.1608
Gradient boosting Classifier	79.8216

CONCLUSION

We have applied various models including: Decision Trees, Gaussian NB, Linear SVC, Polynomial SVC,K means, Gradient Boosting Classifier and Random Forests. We also performed the 10 fold cross validation. The results are different and we have plotted results based on the metrics for the different models. It can further be tested on many models to identify the best that can be used to predict the crime rate. This paper concludes with Decision tree Classifier giving the most balanced results with respect to accuracy, precision, recall and F1 scoreout of three models for prediction of ‘Per Capita Violent Crimes’ feature.
Reduction of overfitting using cross validation improves performance by enough training and testing samples that seemed to help in this analysis by giving correct and consistent performance measures. These predicted features will be useful for the Police Department to utilize their resources

efficiently and take appropriate actions to reduce criminal activities in the society. This can be used to enhance security and protection of criminal data by a desktop or a mobile application to track the crime rate and take any safety measures based on the relevant features.

ACKNOWLEDGEMENT:
We would like to express our sincere gratitude to all those who have been instrumental in the presentation of this project. We are also extremely thankful to Dr Bhargavi R,Vellore Institute of Technology ,Chennai , without her guidance , support and help this research would not have been possible.

REFERENCES:

1) S. Kim, P. Joshi, P. S. Kalsi and P. Taheri, "Crime Analysis Through Machine Learning," 2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2018, pp. 415-420, doi: 10.1109/IEMCON.2018.8614828.
2) International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056 Volume: 05 Issue: 09 | Sep 2018 www.irjet.net p-ISSN: 2395-0072 © 2018, IRJET | Impact Factor value: 7.211 | ISO 9001:2008 Certified Journal | Page 1037 Crime Prediction and Analysis Using Machine Learning Alkesh Bharati1, Dr Sarvanaguru RA.K2
3) Sathyadevan, S., & Gangadharan, S. (2014, August). Crime analysis and prediction using data mining. In Networks & Soft Computing (ICNSC), 2014 First International Conference on (pp. 406-412). IEEE.
4) Yadav, S., Timbadia, M., Yadav, A., Vishwakarma, R., & Yadav, N. (2017, April). Crime pattern detection, analysis & prediction. In Electronics, Communication and AerospaceTechnology (ICECA), 2017 International conference of (Vol. 1, pp. 225- 230).
5) Sivaranjani, S., Sivakumari, S., & Aasha, M. (2016, October). Crime prediction and forecasting in Tamilnadu using clustering approaches. In Emerging Technological Trends (ICETT).