# ECE 657A Assignment 2

July 19, 2021

## 0.1 ECE 657A: Data and Knowledge Modelling and Analysis

### 0.1.1 Assignment 2: Classification using Naive Bayes, Decision Tree, Random Forest, XGBoost

**Group 9 Submission** Ishpinder Kaur i7kaur@uwaterloo.ca
Yuan Sun y228sun@uwaterloo.ca

[63]:
```
# Install necessary packages
!pip install folium
!pip install imbalanced-learn
# NumPy v1.19.5 https://numpy.org/
import numpy as np
from numpy.linalg import eigh
# Pandas v1.3.0 https://pandas.pydata.org/
import pandas as pd
# Matplotlib v3.4.2 https://matplotlib.org/
import matplotlib.pyplot as plt
%matplotlib inline
# seaborn v0.11.1 https://seaborn.pydata.org/
import seaborn as sns
# Folium 0.12.1 http://python-visualization.github.io/folium/
# !pip install folium
import folium
# imblanced 0.8.0 https://imbalanced-learn.org/stable/
from imblearn.over_sampling import SMOTE
# SciPy v1.7.0 https://www.scipy.org/
from scipy.stats import zscore
# scikit-learn v0.24.2 https://scikit-learn.org/
from sklearn import preprocessing
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn import multioutput
from sklearn.naive_bayes import GaussianNB
```

```
# Suppress warning messages for better readability
import warnings
warnings.filterwarnings('ignore')
```

[64]:
```
# Retreiving data from csv file
data= pd.read_csv('dkmacovid_train.csv')
df_train= data.copy()
df_test = pd.read_csv('dkmacovid_kaggletest_features.csv')
```

#### 0.1.2 Data Introduction and Explaination:

**Fields Descriptions:**

- Day: Date in January 2021 ranging from Jan 2 to Jan 31.
- State ID: Arbitrary ID number for each state, based on alphabetical order. Note there are 51 states since the District of Columbia is also included.
- State: Name of the US State.
- Lat: Latitude for the geographic centre of the state.
- Long_: Longitude for the geographic centre of the state.
- Active: Number of active, tracked COVID-19 cases that day in that state.
- Incident_Rate: Incidence Rate = cases per 100,000 persons.
- Total_Test_Results: Total number of people who have been tested.
- Case_Fatality_Ratio: Number recorded deaths * 100/ Number confirmed cases.
- Testing_Rate: Total test results per 100,000 persons. The "total test results" are equal to "Total test results (Positive + Negative)" from COVID Tracking Project.
- Resident Population 2020 Census: The number of population recorded in that State.
- Population Density 2020 Census: Population density is a measure of the average population per square mile of land.
- Density Rank 2020 Census: Density rankings 1 to 52 include the District of Columbia and Puerto Rico.
- SexRatio: the number of males per 100 females. #### Label Description:
- Confirmed: Was there a daily increase in the confirmed total cases of COVID-19 in that state on that day?,if yes, then True else, False
- Deaths: Was there a daily increase in the number of deaths from COVID-19 in that state on that day?,if yes, then True else, False
- Recovered: Was there a daily increase in the number of people recovered from COVID-19 in that state on that day?,if yes, then True else, False

#### 0.1.3 Data Observation:

1) Through info(), the type od data, any null values and their count of row elements is visible. There are three Object data types .
2) Trhough nunique(), their count of unique row elements is visible and also , all the three boject datatypes has same nunique values.
3) Through describe() , invalid values and nature of data can checked.
4) Through isna(), null values can be rechecked.

[65]:
```
# checking for datatypes of each column and its count
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1380 entries, 0 to 1379
Data columns (total 17 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Day                             1380 non-null   int64
 1   State ID                        1380 non-null   int64
 2   State                           1380 non-null   object
 3   Lat                             1380 non-null   float64
 4   Long_                           1380 non-null   float64
 5   Active                          1380 non-null   int64
 6   Incident_Rate                   1380 non-null   float64
 7   Total_Test_Results              1380 non-null   int64
 8   Case_Fatality_Ratio             1380 non-null   float64
 9   Testing_Rate                    1380 non-null   float64
 10  Resident Population 2020 Census 1380 non-null   object
 11  Population Density 2020 Census  1380 non-null   object
 12  Density Rank 2020 Census        1380 non-null   int64
 13  SexRatio                        1380 non-null   int64
 14  Confirmed                       1380 non-null   bool
 15  Deaths                          1380 non-null   bool
 16  Recovered                       1380 non-null   bool
dtypes: bool(3), float64(5), int64(6), object(3)
memory usage: 155.1+ KB
```

[66]:
```
# Checking for unique values
df_train.nunique()
```

[66]:
```
Day                               30
State ID                          46
State                             46
Lat                               46
Long_                             46
Active                          1329
Incident_Rate                   1326
Total_Test_Results              1293
Case_Fatality_Ratio             1326
Testing_Rate                    1293
Resident Population 2020 Census   46
Population Density 2020 Census    46
Density Rank 2020 Census          46
SexRatio                          12
Confirmed                          2
Deaths                             2
Recovered                          2
```

```
dtype: int64
```

[67]:
```
df_train.describe()
```

[67]:
```
                Day     State ID          Lat         Long_        Active  \
count  1380.000000  1380.000000  1380.000000  1380.000000  1.380000e+03
mean     16.500000    25.239130    39.470717   -92.879928  2.610390e+05
std       8.658579    14.513405     6.070494    19.632514  4.914059e+05
min       2.000000     1.000000    21.094300  -157.498300  9.550000e+02
25%       9.000000    12.000000    35.630100  -105.311100  2.731600e+04
50%      16.500000    25.500000    39.583950   -88.259400  1.005915e+05
75%      24.000000    37.000000    43.326600   -77.209800  2.592418e+05
max      31.000000    51.000000    61.370700   -69.381900  3.283336e+06

       Incident_Rate  Total_Test_Results  Case_Fatality_Ratio   Testing_Rate  \
count    1380.000000        1.380000e+03          1380.000000    1380.000000
mean     7203.192905        5.271097e+06             1.631757   91763.237514
std      2305.025102        6.991478e+06             0.656702   40858.185997
min      1232.233261        3.739460e+05             0.439598   30524.071590
25%      6042.134459        1.310515e+06             1.246993   67457.197525
50%      7453.675955        2.919566e+06             1.499993   85438.613770
75%      8621.924085        6.093790e+06             1.817013  104509.453475
max     12811.162350        4.227902e+07             3.928767  235733.711200

       Density Rank 2020 Census     SexRatio
count               1380.000000  1380.000000
mean                  27.173913    97.760870
std                   15.378197     3.219219
min                    1.000000    94.000000
25%                   13.000000    95.000000
50%                   28.500000    97.000000
75%                   41.000000    99.000000
max                   52.000000   109.000000
```

[68]:
```
# Checking for null values
df_train.isna().any()
```

[68]:
```
Day                          False
State ID                     False
State                        False
Lat                          False
Long_                        False
Active                       False
Incident_Rate                False
Total_Test_Results           False
Case_Fatality_Ratio          False
Testing_Rate                 False
```

```
Resident Population 2020 Census    False
Population Density 2020 Census     False
Density Rank 2020 Census           False
SexRatio                           False
Confirmed                          False
Deaths                             False
Recovered                          False
dtype: bool
```

#### 0.1.4 Visualization of areas affected by covid :

The below map shows various areas affected by Covid , mentioned in training data and also, displays the features when clicked.

[69]:
```
# Training Data visualization throup folium maps
def data_plot(df):
    latest_record= df[df['Day']==max(df['Day'])]
    map_plot = folium.Map(location=[33,-102],tiles='Stamen Terrain',min_zoom=1,␣
 ↪max_zoom=8,zoom_start=3.5)
    for i in range(0,len(latest_record)):
        tooltip=  ('<li><bold>State: ' + str(latest_record.iloc[i]['State'])
                   + '<li><bold>Latitude:' + str(latest_record.iloc[i]['Lat'])
                   +'<li><bold>Longitude: ' + str(latest_record.
 ↪iloc[i]['Long_'])
                   +'<li><bold>Active: ' + str(latest_record.iloc[i]['Active'])
                   +'<li><bold>Incident_Rate: ' + str(latest_record.
 ↪iloc[i]['Incident_Rate'])
                   +'<li><bold>Total_Test_Results: ' + str(latest_record.
 ↪iloc[i]['Total_Test_Results'])
                   +'<li><bold>Case_Fatality_Ratio: ' + str(latest_record.
 ↪iloc[i]['Case_Fatality_Ratio'])
                   +'<li><bold>Testing_Rate: ' + str(latest_record.
 ↪iloc[i]['Testing_Rate'])
                   +'<li><bold>Resident Population 2020 Census: '+␣
 ↪str(latest_record.iloc[i]['Resident Population 2020 Census'])
                   +'<li><bold>Population Density 2020 Census: ' +␣
 ↪str(latest_record.iloc[i]['Population Density 2020 Census'])
                   +'<li><bold>Density Rank 2020 Census: ' + str(latest_record.
 ↪iloc[i]['Density Rank 2020 Census'])
                   +'<li><bold>SexRatio: ' + str(latest_record.
 ↪iloc[i]['SexRatio'])
                   +'<li><bold>Confirmed: ' + str(latest_record.
 ↪iloc[i]['Confirmed'])
                   +'<li><bold>Deaths: ' + str(latest_record.iloc[i]['Deaths'])
                   +'<li><bold>Recovered: ' + str(latest_record.
 ↪iloc[i]['Recovered']))
```

```
        folium.Marker(location=[latest_record.iloc[i]['Lat'],latest_record.
↪iloc[i]['Long_']],fill= 'red',
                      tooltip= tooltip,
                      radius= 50).add_to(map_plot)
        folium.CircleMarker(location=[latest_record.
↪iloc[i]['Lat'],latest_record.iloc[i]['Long_']],color='red',fill= 'red',
                      tooltip= tooltip,
                      radius= 10).add_to(map_plot)
    return(map_plot)
# Training Data visualization throup folium maps
display('US States training data records of Covid ')
display(data_plot(data))
```

```
'US States training data records of Covid '
```

```
<folium.folium.Map at 0x16c1cf25040>
```

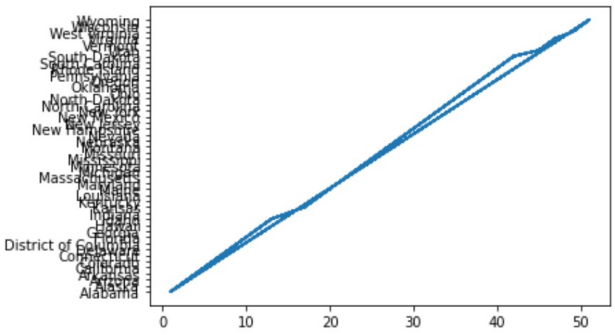
### 0.2 CM[1]:

#### 0.2.1 Data Preprocessing:

1) Through data observation , it is clear that 'Population Density 2020 Census' and 'Resident Population 2020 Census'] is of object type having int type values with string format. So, we will convert them into their appropriate datatype for further analysis.
2) Then, as it is visible that 'State ID' represents the particular 'State' , so, the results can be achieved using one among these two columns. As, 'State' is object datatype ,so, it will be dropped to avoid further labelencoding , and , as 'State ID' is already a numerical value, so, it will be used.
3) Here, the 'Day' column can be treated in various ways:
    a) coversion into datetime timestamp: It can covert the values into datetime format but, in future steps it will make data preprocessing more complex while normalizing. So,It is discarded.
    b) One hot encoding: It can also be coverted into 0,1 format as , each integer here actually represents a particular time category. but, one hot encoding is encouraged in the case of object datatype , as the date categories in int format so, this method is also discarded.
    c) here, date starts from 2 till 31, which represents the change of 1 january to 2 january till 30 of jan to 31 jan.So, the int values here has meaning . So, it should be changed .Example, on day 2 , the congirmed cases are True , that means , the cases have increased compared to 1 january. 1st january here is base , on which the cases of next day is compared. Hence, it is already an integer value and it has special meaning with data. So, it should not be changed or removed.

[70]:
```
plt.plot(df_train['State ID'],df_train['State'])
```

[70]:
```
[<matplotlib.lines.Line2D at 0x16c1f5dbfd0>]
```




[71]:
```
# Function to convert string array to int
def str_int(column):
    list1=[]
    for i in column:
        list1.append(int(i.replace(',','')))
    return list1
# Function to convert string array to float
def str_float(column):
    list2=[]
    for j in column:
        list2.append(float(j.replace(',','')))
    return list2

# Function to return Preprocessed data
def preprocess(x):
    # Conversion of string into float
    if type(x['Population Density 2020 Census'][0]) == str:
        x['Population Density 2020 Census']= str_float(x['Population Density␣
↪2020 Census'])
    # conversion of string into int
    if type(x['Resident Population 2020 Census'][0]) == str:
        x['Resident Population 2020 Census']= str_int(x['Resident Population␣
↪2020 Census'])
    # Dropping 'State' column , as 'State ID' is already present
    if 'State' in x.columns:
        x.drop(['State'],axis=1,inplace=True)
    return x
```

[72]:
```
# Preprocessed data
df_train = preprocess(df_train)
```

#### 0.2.2 Observation of Targets:

**Checking Imbalanced Targets:** Here, there are three targets in this dataset, in which 'Confirmed' is extremely imbalanced , 'Death' column is also unbalanced whereas , 'Recovered' target is balanced fine. The below observation will show how greatly the individual targets are imbalanced . It also displays the need of resampling data.

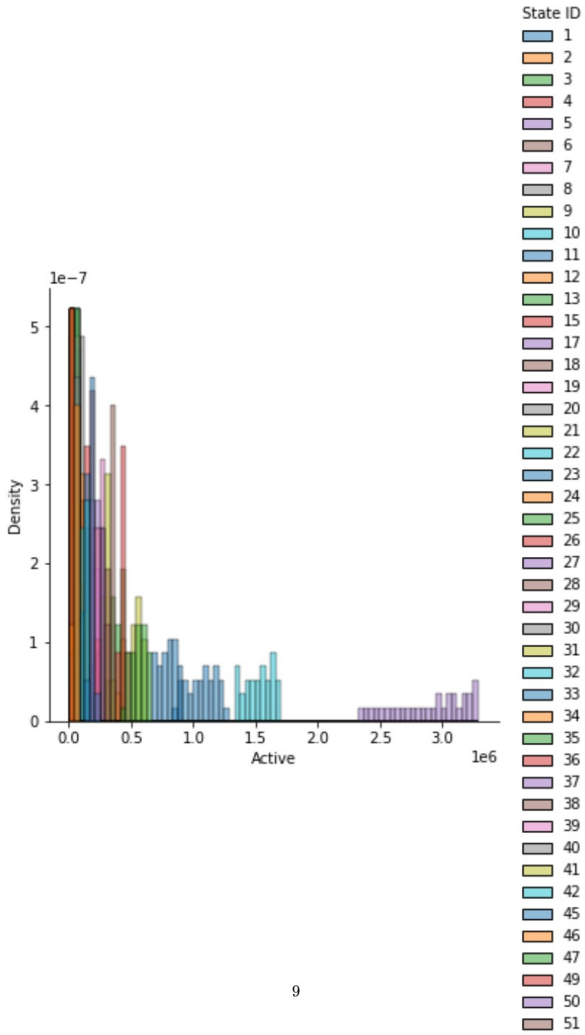
[73]:
```
df_Confirmed_1=df_train[df_train['Confirmed']==True]
df_Confirmed_0=df_train[df_train['Confirmed']==False]
df_Deaths_1=df_train[df_train['Deaths']==True]
df_Deaths_0=df_train[df_train['Deaths']==False]
df_Recovered_1=df_train[df_train['Recovered']==True]
df_Recovered_0=df_train[df_train['Recovered']==False]
print('The count of True cases in Confirmed target are {}'.
 ↪format(df_Confirmed_1.shape[0]))
print('The count of False cases in Confirmed target are {}'.
 ↪format(df_Confirmed_0.shape[0]))
print('The count of True cases in Deaths target are {}'.format(df_Deaths_1.
 ↪shape[0] ))
print('The count of False cases in Deaths target are {}'.format(df_Deaths_0.
 ↪shape[0] ))
print('The count of True cases in Recovered target are {}'.
 ↪format(df_Recovered_1.shape[0] ))
print('The count of False cases in Recovered target are {}'.
 ↪format(df_Recovered_0.shape[0] ))
```

```
The count of True cases in Confirmed target are 1329
The count of False cases in Confirmed target are 51
The count of True cases in Deaths target are 1244
The count of False cases in Deaths target are 136
The count of True cases in Recovered target are 864
The count of False cases in Recovered target are 516
```

**Outlier Detection:** The following graphs will represent the outliers in every feature. But, these outliers should not be removed as they belong to particular area( 'State ID' or 'State'). So, As this is a real time data , these outliers are actually displaying number based on particular state. for example, in 'Active' feature , the outlier belongs to 'State ID'= 5 i.e, California state. Similarly, We can see that in each graph all the outliers are of one colour meaning one State . So, these should not be replaced. If there is any valid outlier left, that will be taken care of in normalization part.

[74]:
```
sns.displot(data=df_train, x='Active', stat='density',hue='State␣
 ↪ID',palette=sns.color_palette(n_colors=46))
```

[74]:
```
<seaborn.axisgrid.FacetGrid at 0x16c1f5b62e0>
```

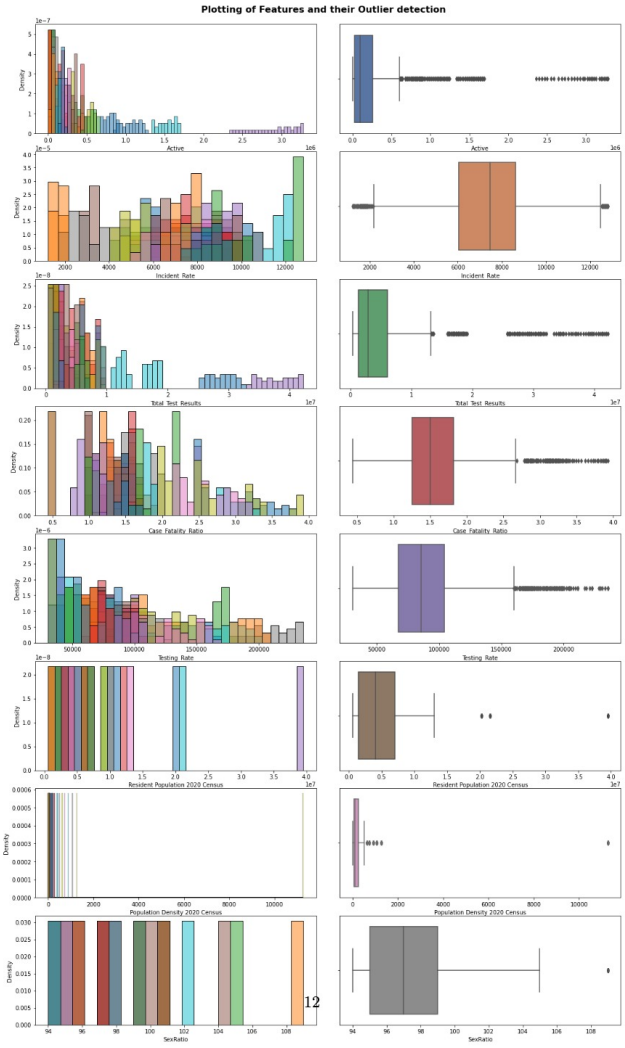

1e−7
5
4
3
2
1
0
Density
0.0
0.5
1.0
1.5
2.0
2.5
3.0
1e6
Active
State ID
1
2
3
4
5
6
7
8
9
10
11
12
13
15
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
45
46
47
49
50
51

9

Below are the graphs of remaining features, here 'Residence population 2020 census' , 'Population densisty 2020 census' and 'sexRatio' show same lengths of bar plots because they, represent the measure of their kind in different states on aparticular day. As there are 46 states , so, is their unique count and any outlier in these graphs is the actual and valid value , varied among states. So, should not be mofified.

[75]:
```
# Outliers detection
color=sns.color_palette('deep',8)
fig, axes= plt.subplots(8,2,figsize=(15,25))
fig.tight_layout()
sns.histplot(data=df_train,x='Active',hue='State ID',stat='density',palette=sns.
 ↪color_palette(n_colors=46),legend=False,ax=axes[0,0])
sns.boxplot(df_train['Active'],color=color[0], ax=axes[0,1])
sns.histplot(data=df_train,x='Incident_Rate',hue='State␣
 ↪ID',stat='density',palette=sns.
 ↪color_palette(n_colors=46),legend=False,ax=axes[1,0])
sns.boxplot(df_train['Incident_Rate'],color=color[1], ax=axes[1,1])
sns.histplot(data=df_train,x='Total_Test_Results',hue='State␣
 ↪ID',stat='density',palette=sns.
 ↪color_palette(n_colors=46),legend=False,ax=axes[2,0])
sns.boxplot(df_train['Total_Test_Results'],color=color[2], ax=axes[2,1])
sns.histplot(data=df_train,x='Case_Fatality_Ratio',hue='State␣
 ↪ID',stat='density',palette=sns.
 ↪color_palette(n_colors=46),legend=False,ax=axes[3,0])
sns.boxplot(df_train['Case_Fatality_Ratio'],color=color[3], ax=axes[3,1])
sns.histplot(data=df_train,x='Testing_Rate',hue='State␣
 ↪ID',stat='density',palette=sns.
 ↪color_palette(n_colors=46),legend=False,ax=axes[4,0])
sns.boxplot(df_train['Testing_Rate'],color=color[4] ,ax=axes[4,1])
sns.histplot(data=df_train,x='Resident Population 2020 Census',hue='State␣
 ↪ID',stat='density',palette=sns.
 ↪color_palette(n_colors=46),legend=False,ax=axes[5,0])
sns.boxplot(df_train['Resident Population 2020 Census'],color=color[5],␣
 ↪ax=axes[5,1])
sns.histplot(data=df_train,x='Population Density 2020 Census',hue='State␣
 ↪ID',stat='density',palette=sns.
 ↪color_palette(n_colors=46),legend=False,ax=axes[6,0])
sns.boxplot(df_train['Population Density 2020 Census'],color=color[6],␣
 ↪ax=axes[6,1])
sns.histplot(data=df_train,x='SexRatio',hue='State␣
 ↪ID',stat='density',palette=sns.
 ↪color_palette(n_colors=46),legend=False,ax=axes[7,0])
sns.boxplot(df_train['SexRatio'],color=color[7], ax=axes[7,1])
```

```
plt.suptitle('Plotting of Features and their Outlier␣
↪detection',fontsize=16,fontweight='bold',y=1.01)
plt.show()
```


Plotting of Features and their Outlier detection
Active
Incident Rate
Total Test Results
Case Fatality Ratio
Testing Rate
Resident Population 2020 Census
Population Density 2020 Census
SexRatio
Density

#### 0.2.3 Train_Test_Split():

Here, data will be assigned proper x_train, x_test, y_train variables for further process.

[76]:
```
# data splitting
x_train = df_train.iloc[:,:-3]
y_train = df_train.iloc[:,-3:]
x_test= df_test.iloc[:,1:]
```

#### 0.2.4 Label Encoding of Targets:

As, the targets are provided in Boolean format, so it will be converted into int (0,1) format for future model fitting without any errors.

[77]:
```
# Label encoding for binary to integer conversion
le= preprocessing.LabelEncoder()
y_train['Confirmed']=le.fit_transform(y_train['Confirmed'])
y_train['Deaths']=le.fit_transform(y_train['Deaths'])
y_train['Recovered']=le.fit_transform(y_train['Recovered'])
```

#### 0.2.5 SMOTE for imbalanced targets:

From, observation of targets , it was clear that 'Confirmed' and 'Death' column are greately imbalanced . So, they will be resampled for achieving individual target predictions and accuracies. Also,'Recovered' target is already balanced . so, we will leave it as it is.Furthermore, their values will be implemented as well for comparision in with and without SMOTE method.

[78]:
```
smote= SMOTE(sampling_strategy='minority')
x_sm_Confirmed, y_sm_Confirmed = smote.fit_resample(x_train,y_train.iloc[:,0])
x_sm_Deaths, y_sm_Deaths = smote.fit_resample(x_train,y_train.iloc[:,1])
y_sm_Confirmed.value_counts() ,y_sm_Deaths.value_counts()
```

[78]:
```
(1    1329
 0    1329
 Name: Confirmed, dtype: int64,
 0    1244
 1    1244
 Name: Deaths, dtype: int64)
```

#### 0.2.6 Normalization :

Normalizarion is used to normalize the data values into a particular range, so, that they provide minimum error and maximum accuracy. Here, we use , zscore normalization, as it automatically handles outliers and provided good results. Normalization will only be applied to x train data. Below, is the defined function to get normalized dataframe of particular data.

[79]:
```
# Normalizing data using Zscore
def norm(df):
    zscore_norm= zscore(df)
    x_train = pd.DataFrame(zscore_norm, columns= df.columns)
    return(x_train)
x_Confirmed = norm(x_sm_Confirmed)
x_Deaths = norm(x_sm_Deaths)
x_Recovered = norm(x_train)
x_train = norm(x_train)
```

### 0.3 CM[2]:

### 0.4 PCA:

- Principal component analysis is a dimension reduction technique which transforms correlated features into less count of uncorrelated one, which are known as principal components.
- It Compresses the size of data and keeps the only and most important components required for predictions.
- The first component is supposed to have largest variance , which will keep decreasing till last principal component.
- for every new added component , the inertia will increase in susbspace and new axis will face less inertia than original compunents(Optimal).
- Therefore, to choose optimal component , variance or inertia should be calculated as, as lower the components , lower is the variance. 95% explained variance is used to compute the optimal number of principle components .
- Although to be sure , one can also the check the performance of number of principal components based on the accuracy they provide.

#### 0.4.1 Finding best number of components or features to be reduced

#### 0.4.2 a) Based on training set accuracy

[80]:
```
# Finding best number of components or features to be reduced w.r.t. accuracy␣
↪of validation data
# Using KNN Classifier

from sklearn.neighbors import KNeighborsClassifier
scores=[]
for i in range(2,14):
    pca=PCA(n_components=i)
    pca.fit(x_train)
    x_pca= pca.transform(x_train)
    knn= KNeighborsClassifier()
    score= cross_val_score(knn,x_pca,y_train,cv=5, scoring='f1_macro')
    scores.append(score.mean())
plt.figure(figsize=(15,3))
plt.plot(np.arange(2,14),scores,marker='o',markerfacecolor='r',color='g')
plt.xlabel("n_components in PCA")
```

[79]:
```
# Normalizing data using Zscore
def norm(df):
    zscore_norm= zscore(df)
    x_train = pd.DataFrame(zscore_norm, columns= df.columns)
    return(x_train)
x_Confirmed = norm(x_sm_Confirmed)
x_Deaths = norm(x_sm_Deaths)
x_Recovered = norm(x_train)
x_train = norm(x_train)
```

### 0.3 CM[2]:

### 0.4 PCA:

- Principal component analysis is a dimension reduction technique which transforms correlated features into less count of uncorrelated one, which are known as principal components.
- It Compresses the size of data and keeps the only and most important components required for predictions.
- The first component is supposed to have largest variance , which will keep decreasing till last principal component.
- for every new added component , the inertia will increase in susbspace and new axis will face less inertia than original compunents(Optimal).
- Therefore, to choose optimal component , variance or inertia should be calculated as, as lower the components , lower is the variance. 95% explained variance is used to compute the optimal number of principle components .
- Although to be sure , one can also the check the performance of number of principal components based on the accuracy they provide.

#### 0.4.1 Finding best number of components or features to be reduced

#### 0.4.2 a) Based on training set accuracy

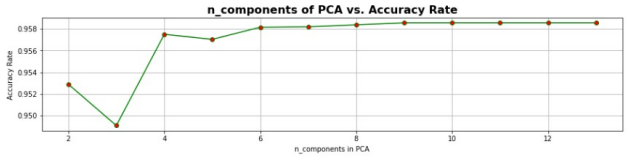
[80]:
```
# Finding best number of components or features to be reduced w.r.t. accuracy␣
↪of validation data
# Using KNN Classifier

from sklearn.neighbors import KNeighborsClassifier
scores=[]
for i in range(2,14):
    pca=PCA(n_components=i)
    pca.fit(x_train)
    x_pca= pca.transform(x_train)
    knn= KNeighborsClassifier()
    score= cross_val_score(knn,x_pca,y_train,cv=5, scoring='f1_macro')
    scores.append(score.mean())
plt.figure(figsize=(15,3))
plt.plot(np.arange(2,14),scores,marker='o',markerfacecolor='r',color='g')
plt.xlabel("n_components in PCA")
```

```
plt.ylabel("Accuracy Rate")
plt.title("n_components of PCA vs. Accuracy Rate", fontsize=16,␣
 ↪fontweight='bold')
plt.grid()
scores= pd.DataFrame(scores, columns= ['accuracy'])
scores['n_components']= np.arange(2,14)
print('Accuracy rate of varying n_components of PCA: ')
display(scores.T)
```

```
Accuracy rate of varying n_components of PCA:
```

| | 0 | 1 | 2 | 3 | 4 | 5 | \ |
|---|---|---|---|---|---|---|---|
| accuracy | 0.952896 | 0.949078 | 0.9575 | 0.957035 | 0.958151 | 0.958194 | |
| n_components | 2.000000 | 3.000000 | 4.0000 | 5.000000 | 6.000000 | 7.000000 | |

| | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|
| accuracy | 0.958378 | 0.958562 | 0.958562 | 0.958562 | 0.958562 | 0.958562 |
| n_components | 8.000000 | 9.000000 | 10.000000 | 11.000000 | 12.000000 | 13.000000 |



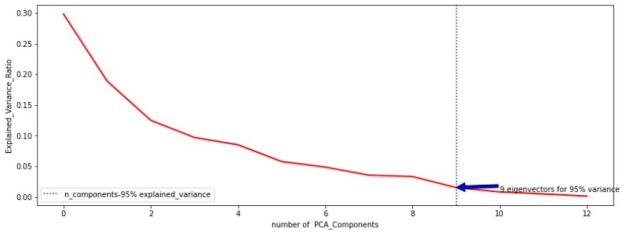

### 0.4.3 b) Based on Explained_Variance_

95% of explained_variance_ is checked, whenever to tune the n_components parameter. Here, 95% of variance of data including all features is 12.35 . So, the first n_component providing a value higher than that will be treated as optimal component.

[81]:
```
# first, the PCA of original datset will be calculated
num=x_train.shape[1]
pca=PCA(n_components=num)
x_pca= pca.fit(x_train).transform(x_train)
# calculation of original dataset variance
variance_sum=sum(pca.explained_variance_)
print('Total variance present in dataset is: {}'.format(variance_sum))
# Calculation of 95% variance
variance_95= variance_sum*0.95
print('The required 95% variance is: {}'.format(variance_95))

# Tuning the best n_components for PCA
```

```
pca_df= zip (range(0,num),pca.explained_variance_)
pca_df= pd.DataFrame(pca_df, columns=['PCA_Component','Explained_Variance'])
for i in range(1,num):
    print('Explained Variance w.r.t. {} PCA Components: {}'.
 ↪format(i,sum(pca_df['Explained_Variance'][0:i])))

# Plotting graph of data
plt.figure(1,figsize=(14,5))
plt.plot(pca.explained_variance_ratio_,lw=2,c='r')
plt.xlabel("number of  PCA_Components")
plt.ylabel("Explained_Variance_Ratio")
# Plotting 95% explained_variance line
plt.axvline(9, linestyle=':',label='n_components-95%␣
 ↪explained_variance',c='blue')
plt.legend()
# adding arrow to point the best found n_components
plt.annotate('9 eigenvectors for 95% variance',xy=(9,pca.
 ↪explained_variance_ratio_[9]),
             xytext=(10,pca.
 ↪explained_variance_ratio_[10]),arrowprops=dict(facecolor='blue'))
plt.show()
```

```
Total variance present in dataset is: 13.009427121102263
The required 95% variance is: 12.358955765047149
Explained Variance w.r.t. 1 PCA Components: 3.879785531551403
Explained Variance w.r.t. 2 PCA Components: 6.3363849544202475
Explained Variance w.r.t. 3 PCA Components: 7.962147802805233
Explained Variance w.r.t. 4 PCA Components: 9.2256539531502
Explained Variance w.r.t. 5 PCA Components: 10.33300253579973
Explained Variance w.r.t. 6 PCA Components: 11.084514399347634
Explained Variance w.r.t. 7 PCA Components: 11.71880606810373
Explained Variance w.r.t. 8 PCA Components: 12.182053175850294
Explained Variance w.r.t. 9 PCA Components: 12.615995368416371
Explained Variance w.r.t. 10 PCA Components: 12.816763993533424
Explained Variance w.r.t. 11 PCA Components: 12.92505180314468
Explained Variance w.r.t. 12 PCA Components: 12.99067317349017
```




#### 0.4.4 Applying best found n_components:

The best training data accuracy is found at number of components= 9 as well as, through explained_variance also, 9 turns out to be the best

[82]:
```
# Function that returns PCA of given data and number of components
def pca(df,n):
    pca_=PCA(n_components=n)
    pca_1= pca_.fit(df).transform(df)
    return(pca_1)
# PCA dataframe with best found n_components
pca_best= pca(x_train,9)
```

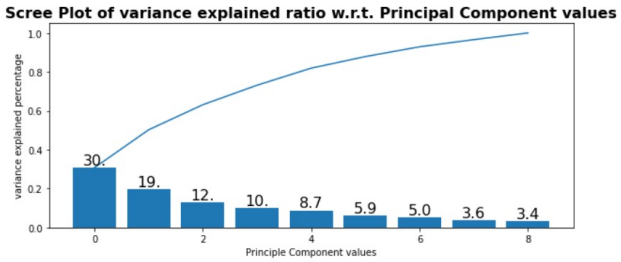
#### 0.4.5 Scree Plot with best Found parameter on PCA :

The below plot is showing that as the number of components increases their variance will decrease , although cumulative variance might increase. the first PCA component has large variance among all.

[83]:
```
# to calculate explained variance ratio
egnvalues, egnvectors = eigh(np.cov(pca_best,rowvar=False))
explained_var_ratio=[]
for i in sorted(egnvalues, reverse=True):
    explained_var_ratio.append((i/sum(egnvalues)))

# values consisting of explained_variance_ratio_ of PCA and respective index
values = explained_var_ratio
index= np.arange(len(values))

# Plotting scree plot
plt.figure(figsize=(10,4))
cumulative_variance= np.cumsum(values)
plt.bar(index,values)
```

```
plt.plot(index,cumulative_variance)
for i in range(len(values)):
    plt.annotate(r'%s' % ((str(values[i]*100)[:3])),(index[i],values[i]),␣
 ↪va='bottom',ha='center', fontsize=16)
plt.xlabel('Principle Component values')
plt.ylabel('variance explained percentage')
plt.title('Scree Plot of variance explained ratio w.r.t. Principal Component␣
 ↪values',fontsize=16,fontweight='bold')
```

[83]:
```
Text(0.5, 1.0, 'Scree Plot of variance explained ratio w.r.t. Principal
Component values')
```




[84]:
```
cumulative_variance
```

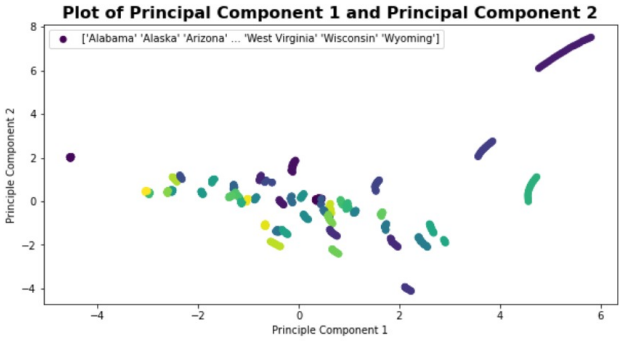
[84]:
```
array([0.30752909, 0.5022501 , 0.63111531, 0.73126643, 0.81903982,
       0.878608  , 0.92888478, 0.96560381, 1.        ])
```

#### 0.4.6 Plotting of PCA1 and PCA2 data elemnets :

The below plot will display the first two principal components (important ones) and their distribution among different state feature.

[85]:
```
plt.figure(figsize=(10,5))
plt.scatter(pca_best[:,0],pca_best[:,1],c= x_train['State␣
 ↪ID'],label=data['State'].values)
plt.xlabel('Principle Component 1')
plt.ylabel('Principle Component 2')
plt.legend(loc='best')
plt.title('Plot of Principal Component 1 and Principal Component␣
 ↪2',fontsize=16,fontweight='bold')
```

[85]: Text(0.5, 1.0, 'Plot of Principal Component 1 and Principal Component 2')




### 0.5 CM[2]:

#### 0.5.1 LDA

Linear Discriminant Analysis is also dimensionality reduction technique. It is a supervised classification technique used in preprocessing and implemented for competitive machine learning models.

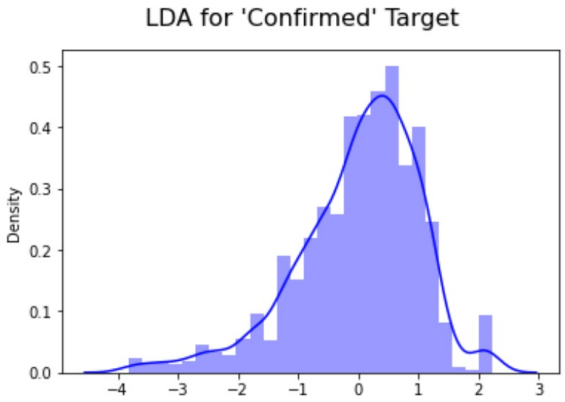
Yes, LDA provides different performance results for different targets . For example, around 96% accuracy in 'Confirmed' target , 89% in 'Deaths' target and 77% in 'Recovered target . This is due to different balancing of targets. As, most of the targets of 'Confirmed' and 'Death' are True , and in case of 'Recovered' partial of the data is True and False.

**LDA with 'Confirmed' Target output**

[86]:
```
# Applying LDA on 'Confirmed' target
lda= LDA(n_components=1)
X1= lda.fit(x_train,y_train.iloc[:,0]).transform(x_train)
lda.fit(x_train,y_train.iloc[:,0])
lda_pred1=lda.predict(x_train)
lda_score1=accuracy_score(y_train.iloc[:,0],lda_pred1)
print("The accuracy score for 'Confirmed' target by LDA: {}".format(lda_score1))
sns.distplot(X1,color='blue')
plt.suptitle("LDA for 'Confirmed' Target", fontsize=16)
```

```
The accuracy score for 'Confirmed' target by LDA: 0.9601449275362319
```

[86]: Text(0.5, 0.98, "LDA for 'Confirmed' Target")

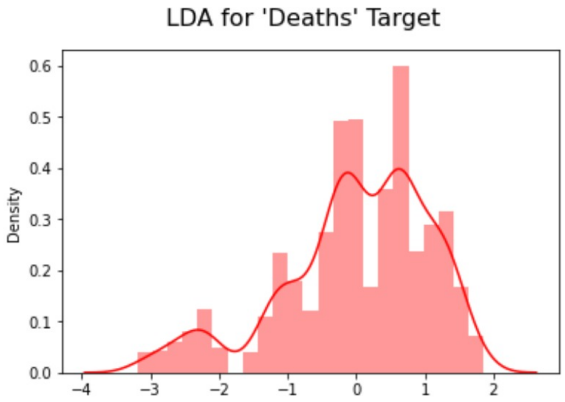



**LDA with 'Deaths' Target output**

[87]:
```
# Applying LDA on 'Deaths' target
X2= lda.fit(x_train,y_train.iloc[:,1]).transform(x_train)
lda.fit(x_train,y_train.iloc[:,1])
lda_pred2=lda.predict(x_train)
lda_score2=accuracy_score(y_train.iloc[:,1],lda_pred2)
print("The accuracy score for 'Deaths' target by LDA: {}".format(lda_score2))
sns.distplot(X2,color='red')
plt.suptitle("LDA for 'Deaths' Target", fontsize=16)
```

```
The accuracy score for 'Deaths' target by LDA: 0.8905797101449275
```

[87]:
```
Text(0.5, 0.98, "LDA for 'Deaths' Target")
```

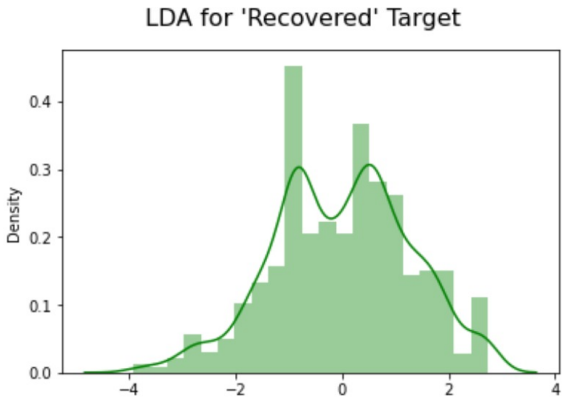



**LDA with 'Recovered' Target output**

[88]:
```
# Applying LDA on 'Recovered' target
X3= lda.fit(x_train,y_train.iloc[:,2]).transform(x_train)
lda.fit(x_train,y_train.iloc[:,2])
lda_pred3=lda.predict(x_train)
lda_score3=accuracy_score(y_train.iloc[:,2],lda_pred3)
print("The accuracy score for 'Recovered' target by LDA: {}".format(lda_score3))
sns.distplot(X3,color='green')
plt.suptitle("LDA for 'Recovered' Target", fontsize=16)
```

```
The accuracy score for 'Recovered' target by LDA: 0.7746376811594203
```

[88]:
```
Text(0.5, 0.98, "LDA for 'Recovered' Target")
```




Here, LDA with 'Confirmed' target shows maximum accuracy on training data and 'Recovered' target gave minimum accuracy among all.

### 0.6 Hyperparameter tuning with K-Fold Cross Validation:

This is used for further all parts.

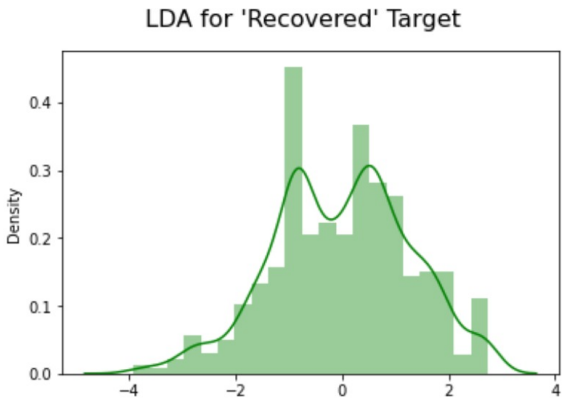
[89]:
```
# Hyperparameter tuning with K-Fold Crossvalidation, using GridSearchCV()
def hyperparameter_tuning(model,params,train,test):
    clf=GridSearchCV(model,params,cv=5,return_train_score=False)
    clf.fit(train,test)
    df=pd.DataFrame(clf.cv_results_)
    return df
```

### 0.7 CM[3]:

#### 0.7.1 Decision Tree Classifier:

1) With Validation:
   - Without SMOTE

   a) Best max_depth for Combined targets is 5
   b) Best max_depth for 'Confirmed' targets is 3
   c) Best max_depth for 'Deaths' target is 3
   d) Best max_depth for 'Recovered' target is 10
   - With SMOTE




Here, LDA with 'Confirmed' target shows maximum accuracy on training data and 'Recovered' target gave minimum accuracy among all.

### 0.6 Hyperparameter tuning with K-Fold Cross Validation:

This is used for further all parts.

[89]:
```
# Hyperparameter tuning with K-Fold Crossvalidation, using GridSearchCV()
def hyperparameter_tuning(model,params,train,test):
    clf=GridSearchCV(model,params,cv=5,return_train_score=False)
    clf.fit(train,test)
    df=pd.DataFrame(clf.cv_results_)
    return df
```

### 0.7 CM[3]:

#### 0.7.1 Decision Tree Classifier:

1) With Validation:
   - Without SMOTE
   a) Best max_depth for Combined targets is 5
   b) Best max_depth for 'Confirmed' targets is 3
   c) Best max_depth for 'Deaths' target is 3
   d) Best max_depth for 'Recovered' target is 10
   - With SMOTE

a) Best max_depth for 'Confirmed' targets is 3
b) Best max_depth for 'Deaths' target is 5

2) Without Validation:
   - Without SMOTE
   a) Best max_depth for Combined targets is None
   b) Best max_depth for 'Confirmed' targets is None
   c) Best max_depth for 'Deaths' target is None
   d) Best max_depth for 'Recovered' target is None
   - With SMOTE
   a) Best max_depth for 'Confirmed' targets is None
   b) Best max_depth for 'Deaths' target is None
3) The splitting rules , also known as criterion parameter is defined as gini and entropy. Among which gini provided slightly better results than entropy on some different max_depths. So, as the targets are imbalanced , thus, gini and entropy handles it fine,but not really good. Here, gini impurity lies in between 0 to 0.5 , whereas as entropy impurity lies in between 0 to 1. Thus, gini performs better for best feature selection.

So, max_depth here treats every target differently. Thus, their combination may provide better results.

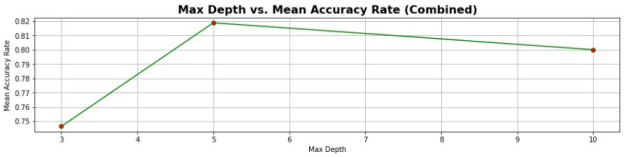
#### 0.7.2 Hyper Parameter tuning of max_depth(With K-fold Cross Validation):

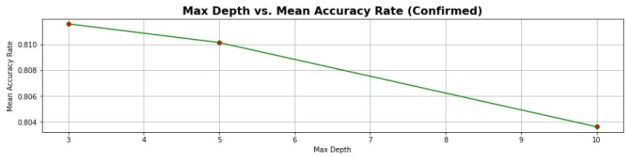
[90]:
```
def dtc_With_Validation(x,y,str1):
    # initiating DecisionTreeClassifier() and setting hyperparameters
    dtc=DecisionTreeClassifier()
    parameters1={'max_depth':[3,5,10,None]}

    # Decision Tree Classifier using 5-fold cross validation on original data
    dtc_df=hyperparameter_tuning(dtc,parameters1,x,y)
    dtc_df=dtc_df[['param_max_depth','mean_test_score']]

    # Plotting graph of Max Depth vs. Accuracy Rate
    plt.figure(figsize=(15,3))
    plt.
↪plot(dtc_df['param_max_depth'],dtc_df['mean_test_score'],marker='o',markerfacecolor='r',col
    plt.xlabel("Max Depth")
    plt.ylabel("Mean Accuracy Rate")
    plt.title("Max Depth vs. Mean Accuracy Rate ({})".format(str1),␣
↪fontsize=16, fontweight='bold')
    plt.grid()

    # Printing exact values
    print("Accuracies w.r.t. varying Max depth, using original features({})".
↪format(str1))
    display(dtc_df)
```

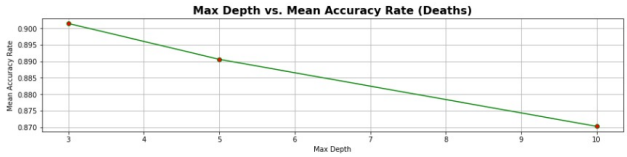
### 0.7.3 Without SMOTE:

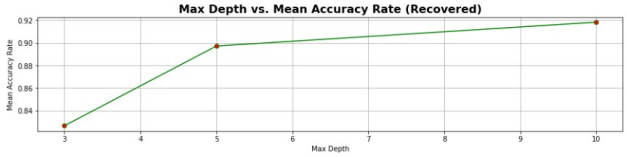
[91]:
```
dtc_With_Validation_Combined= dtc_With_Validation(x_train,y_train,'Combined')
dtc_With_Validation_Confirmed=␣
 ↪dtc_With_Validation(x_train,y_train['Confirmed'],'Confirmed')
dtc_With_Validation_Deaths=␣
 ↪dtc_With_Validation(x_train,y_train['Deaths'],'Deaths')
dtc_With_Validation_Recovered=␣
 ↪dtc_With_Validation(x_train,y_train['Recovered'],'Recovered')
```

```
Accuracies w.r.t. varying Max depth, using original features(Combined)

  param_max_depth  mean_test_score
0               3         0.746377
1               5         0.818841
2              10         0.800000
3            None         0.777536

Accuracies w.r.t. varying Max depth, using original features(Confirmed)

  param_max_depth  mean_test_score
0               3         0.811594
1               5         0.810145
2              10         0.803623
3            None         0.797826

Accuracies w.r.t. varying Max depth, using original features(Deaths)

  param_max_depth  mean_test_score
0               3         0.901449
1               5         0.890580
2              10         0.870290
3            None         0.874638

Accuracies w.r.t. varying Max depth, using original features(Recovered)

  param_max_depth  mean_test_score
0               3         0.826812
1               5         0.897101
2              10         0.918116
3            None         0.921014
```





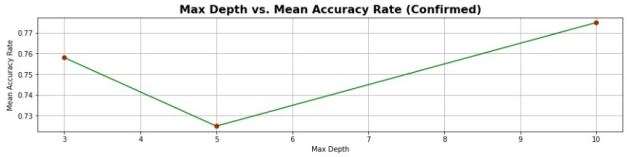

#### 0.7.4 With SMOTE:

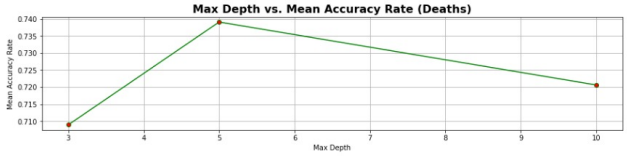
[92]:
```
dtc_With_Validation_Confirmed_SMOTE=␣
↪dtc_With_Validation(x_Confirmed,y_sm_Confirmed,'Confirmed')
dtc_With_Validation_Deaths_SMOTE=␣
↪dtc_With_Validation(x_Deaths,y_sm_Deaths,'Deaths')
```

```
Accuracies w.r.t. varying Max depth, using original features(Confirmed)

  param_max_depth  mean_test_score
0               3         0.757978
1               5         0.724942
2              10         0.774966
3            None         0.756912
```

```
Accuracies w.r.t. varying Max depth, using original features(Deaths)
```

| | param_max_depth | mean_test_score |
|---|---|---|
| 0 | 3 | 0.708915 |
| 1 | 5 | 0.739105 |
| 2 | 10 | 0.720618 |
| 3 | None | 0.704141 |




#### 0.7.5 Hyper Parameter tuning(max_depth) without validation:

[93]:
```
def dtc_Without_Validation(x,y,str1):
    # accuracies using training set without validation
    scores=[]
    maxDepth=[3,5,10,None]
    for i in maxDepth:
        tree= DecisionTreeClassifier(max_depth=i).fit(x,y)
        scores.append(accuracy_score(y,tree.predict(x)))
    # Plotting graph of Max Depth vs. Accuracy Rate without validation
    plt.figure(figsize=(15,3))
    plt.plot(maxDepth,scores,marker='o',markerfacecolor='r',color='g')
    plt.xlabel("Max Depth")
    plt.ylabel("Mean Accuracy Rate")
    plt.title("Max Depth vs. Mean Accuracy Rate without validation({})".
↪format(str1), fontsize=16, fontweight='bold')
    plt.grid()
```

```
    scores=pd.
↪DataFrame(scores,index=["Max_depth=3","Max_depth=5","Max_depth=10","Max_depth=None"],column

    # Printing exact accuracies
    print('The accuracy scores without K-Fold CrossValidation are({})'.
↪format(str1))
    display(scores)
```

#### 0.7.6 Without SMOTE:

[94]:
```
dtc_Without_Validation_Combined=␣
↪dtc_Without_Validation(x_train,y_train,'Combined')
dtc_Without_Validation_Confirmed=␣
↪dtc_Without_Validation(x_train,y_train['Confirmed'],'Confirmed')
dtc_Without_Validation_Deaths=␣
↪dtc_Without_Validation(x_train,y_train['Deaths'],'Deaths')
dtc_Without_Validation_Recovered=␣
↪dtc_Without_Validation(x_train,y_train['Recovered'],'Recovered')
```

```
The accuracy scores without K-Fold CrossValidation are(Combined)
```

| | Training_Set_Accuracy |
|---|---|
| Max_depth=3 | 0.751449 |
| Max_depth=5 | 0.838406 |
| Max_depth=10 | 0.936232 |
| Max_depth=None | 1.000000 |

```
The accuracy scores without K-Fold CrossValidation are(Confirmed)
```

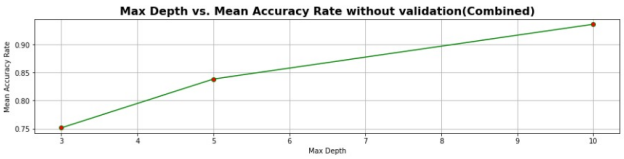
| | Training_Set_Accuracy |
|---|---|
| Max_depth=3 | 0.965942 |
| Max_depth=5 | 0.971739 |
| Max_depth=10 | 0.988406 |
| Max_depth=None | 1.000000 |

```
The accuracy scores without K-Fold CrossValidation are(Deaths)
```

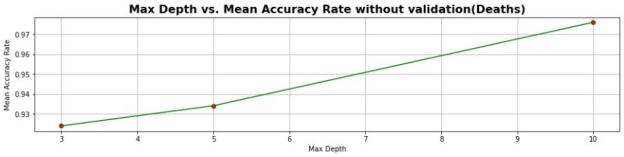
| | Training_Set_Accuracy |
|---|---|
| Max_depth=3 | 0.923913 |
| Max_depth=5 | 0.934058 |
| Max_depth=10 | 0.976087 |
| Max_depth=None | 1.000000 |

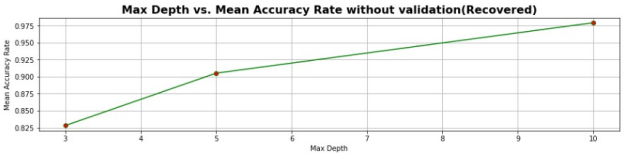
```
The accuracy scores without K-Fold CrossValidation are(Recovered)
```

| | Training_Set_Accuracy |
|---|---|
| Max_depth=3 | 0.828261 |
| Max_depth=5 | 0.905072 |
| Max_depth=10 | 0.978986 |
| Max_depth=None | 1.000000 |


Max Depth vs. Mean Accuracy Rate without validation(Combined)
Mean Accuracy Rate
0.90
0.85
0.80
0.75
3
4
5
6
7
8
9
10
Max Depth


Max Depth vs. Mean Accuracy Rate without validation(Confirmed)
Mean Accuracy Rate
0.985
0.980
0.975
0.970
0.965
3
4
5
6
7
8
9
10
Max Depth

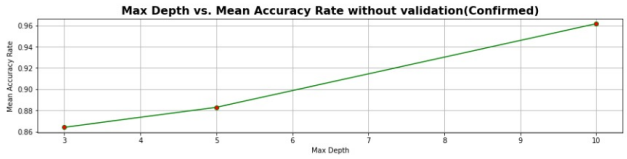

Max Depth vs. Mean Accuracy Rate without validation(Deaths)
Mean Accuracy Rate
0.97
0.96
0.95
0.94
0.93
3
4
5
6
7
8
9
10
Max Depth


Max Depth vs. Mean Accuracy Rate without validation(Recovered)
Mean Accuracy Rate
0.975
0.950
0.925
0.900
0.875
0.850
0.825
3
4
5
6
7
8
9
10
Max Depth

#### 0.7.7 With SMOTE:

[95]:
```
dtc_Without_Validation_Confirmed_SMOTE=␣
↪dtc_Without_Validation(x_Confirmed,y_sm_Confirmed,'Confirmed')
dtc_Without_Validation_Deaths_SMOTE=␣
↪dtc_Without_Validation(x_Deaths,y_sm_Deaths,'Deaths')
```

```
The accuracy scores without K-Fold CrossValidation are(Confirmed)

              Training_Set_Accuracy
Max_depth=3                0.864184
Max_depth=5                0.882995
Max_depth=10               0.961625
Max_depth=None             1.000000

The accuracy scores without K-Fold CrossValidation are(Deaths)

              Training_Set_Accuracy
Max_depth=3                0.807074
Max_depth=5                0.871785
Max_depth=10               0.941318
Max_depth=None             1.000000
```





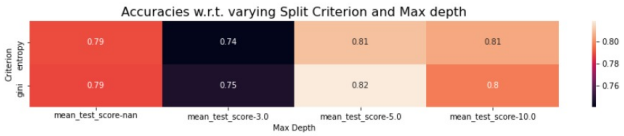
Here, in validation set , the maximum accuracy is reached at 5 whereas , on actual training data set 100% accuracy is achieved at default parameter None i.e., grow until leaf contains 2 elements.

#### 0.7.8 Effect on Accuracies w.r.t. Split method(criterion) and Max Depth :

[96]:
```
# setting parameters for hyperparameter tuning of criterion and max depth
dtc=DecisionTreeClassifier()
parameters1_={'criterion':['gini','entropy'],'max_depth':[3,5,10,None]}
dtc_df_=hyperparameter_tuning(dtc,parameters1_,x_train,y_train)
dtc_df_=dtc_df_[['param_criterion','param_max_depth','mean_test_score']]
dtc_df_=dtc_df_.pivot(index='param_criterion',columns='param_max_depth')

# Plotting Heatmap of Split method and Max Depth w.r.t. accuracies
plt.figure(figsize=(15,2))
sns.heatmap(dtc_df_,annot=True)
plt.xlabel("Max Depth")
plt.ylabel("Criterion")
plt.title("Accuracies w.r.t. varying Split Criterion and Max depth",fontsize=16)

# Printing exact accuracies
print('accuracies of varying Split Criterion and Max depth are: ')
display(dtc_df_)
```

```
accuracies of varying Split Criterion and Max depth are:
```

| | mean_test_score | | | |
|---|---|---|---|---|
| param_max_depth | NaN | 3 | 5 | 10 |
| param_criterion | | | | |
| entropy | 0.789130 | 0.740580 | 0.809420 | 0.805072 |
| gini | 0.788406 | 0.746377 | 0.818116 | 0.797101 |




### 0.8 CM[4]:

#### 0.8.1 Random Forest Classifier

1) With Validation:
   - Without SMOTE

a) Best max_depth and n_estimators for Combined targets are 5 and 200 respectively

b) Best max_depth and n_estimators for 'Confirmed' targets are 5 and 200 respectively

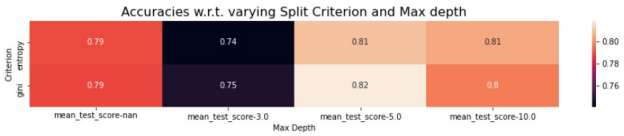
c) Best max_depth and n_estimators for 'Deaths' target are None and 50 respectively

#### 0.7.8 Effect on Accuracies w.r.t. Split method(criterion) and Max Depth :

[96]:
```
# setting parameters for hyperparameter tuning of criterion and max depth
dtc=DecisionTreeClassifier()
parameters1_={'criterion':['gini','entropy'],'max_depth':[3,5,10,None]}
dtc_df_=hyperparameter_tuning(dtc,parameters1_,x_train,y_train)
dtc_df_=dtc_df_[['param_criterion','param_max_depth','mean_test_score']]
dtc_df_=dtc_df_.pivot(index='param_criterion',columns='param_max_depth')

# Plotting Heatmap of Split method and Max Depth w.r.t. accuracies
plt.figure(figsize=(15,2))
sns.heatmap(dtc_df_,annot=True)
plt.xlabel("Max Depth")
plt.ylabel("Criterion")
plt.title("Accuracies w.r.t. varying Split Criterion and Max depth",fontsize=16)

# Printing exact accuracies
print('accuracies of varying Split Criterion and Max depth are: ')
display(dtc_df_)
```

```
accuracies of varying Split Criterion and Max depth are:
```

| | mean_test_score | | | |
|---|---|---|---|---|
| param_max_depth | NaN | 3 | 5 | 10 |
| param_criterion | | | | |
| entropy | 0.789130 | 0.740580 | 0.809420 | 0.805072 |
| gini | 0.788406 | 0.746377 | 0.818116 | 0.797101 |




### 0.8 CM[4]:

#### 0.8.1 Random Forest Classifier

1) With Validation:
   - Without SMOTE

a) Best max_depth and n_estimators for Combined targets are 5 and 200 respectively

b) Best max_depth and n_estimators for 'Confirmed' targets are 5 and 200 respectively

c) Best max_depth and n_estimators for 'Deaths' target are None and 50 respectively

d) Best max_depth and n_estimators for 'Recovered' target are 3 and 200 respectively

- With SMOTE

a) Best max_depth and n_estimators for 'Confirmed' targets are 10 and 200 respectively

b) Best max_depth and n_estimators for 'Deaths' target are 5 and 50 respectively

2) Without Validation:

- Without SMOTE

a) Best max_depth and n_estimators for Combined targets are None and 200 respectively

b) Best max_depth and n_estimators for 'Confirmed' targets are None and 200 respectively

c) Best max_depth and n_estimators for 'Deaths' target are None and 200 respectively

d) Best max_depth and n_estimators for 'Recovered' target are None and 200 respectively

- With SMOTE

a) Best max_depth and n_estimators for 'Confirmed' targets are None and 200 respectively

b) Best max_depth and n_estimators for 'Deaths' target are None and 200 respectively

3) PCA Features:

a)With Validation: Best max_depth and n_estimators for targets are 10 and 200 respectively
b)Without Validation: Best max_depth and n_estimators for targets are None and 200 respectively

So, best n_estimators among all results are 200 and max_depth varies in acse of k-fold validation.here, In original max_depth=5 is good whereas, in PCA max_depth=10 is good. Although, without validation max_depth = None provides 100% accuracy.

#### 0.8.2 Hyperparameter tuning with K-fold Cross Validation:

[97]:
```
# Function that returns RandomForestClassifier accuracy with kfold cross␣
↪validation
def rfc_With_validation(x,y,str1,str2):
    # initiating RandomForestClassifier() and setting hyperparameters
    rfc=RandomForestClassifier()
    parameters2={'n_estimators': [5,10,50,150,200],'max_depth': [3,5,10,None]}
    # Random Forest Classifier with 5-fold crossvalidation
    rfc_df=hyperparameter_tuning(rfc,parameters2,x,y)
    rfc_df=rfc_df[['param_n_estimators','param_max_depth','mean_test_score']]
    rfc_df=rfc_df.pivot(index='param_n_estimators',columns='param_max_depth')
    # Plotting Heatmap of accuracies achieved
    plt.figure(figsize=(15,4))
    sns.heatmap(rfc_df,annot=True)
    plt.xlabel("Max Depth")
    plt.ylabel("No. of Trees")
    plt.title("Accuracies w.r.t. varying No. of trees and Max depth,\n using {}␣
↪features and {} target(With K-fold Cross Validation)".format(str1,str2)
```

```
            ,fontsize=16)
    # Printing exact accuracies
    print('accuracies of varying No. of trees and Max depth, using {} features␣
↪and {} target(With K-fold Cross Validation) are:'.format(str1,str2))
    display(rfc_df)
```

#### 0.8.3 Hyper Parameter tuning of No. of trees and max_depth using original features(With Validation):

#### 0.8.4 Without SMOTE:

[98]:
```
rfc_With_Validation_Original_Combined=rfc_With_validation(x_train,y_train,'Original','Combined
rfc_With_Validation_Original_Confirmed=rfc_With_validation(x_train,y_train['Confirmed'],'Origi
rfc_With_Validation_Original_Deaths=rfc_With_validation(x_train,y_train['Deaths'],'Original','
rfc_With_Validation_Original_Recovered=rfc_With_validation(x_train,y_train['Recovered'],'Origi
```

```
accuracies of varying No. of trees and Max depth, using Original features and
Combined target(With K-fold Cross Validation) are:
```

| | mean_test_score | | | |
|---|---|---|---|---|
| param_max_depth | NaN | 3 | 5 | 10 |
| param_n_estimators | | | | |
| 5 | 0.806522 | 0.765217 | 0.831159 | 0.814493 |
| 10 | 0.822464 | 0.785507 | 0.828261 | 0.828986 |
| 50 | 0.818841 | 0.804348 | 0.832609 | 0.832609 |
| 150 | 0.819565 | 0.786957 | 0.839130 | 0.826087 |
| 200 | 0.819565 | 0.794928 | 0.839130 | 0.835507 |

```
accuracies of varying No. of trees and Max depth, using Original features and
Confirmed target(With K-fold Cross Validation) are:
```

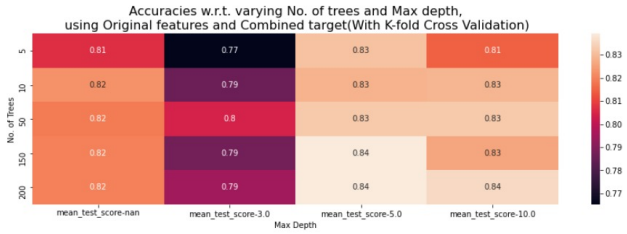
| | mean_test_score | | | |
|---|---|---|---|---|
| param_max_depth | NaN | 3 | 5 | 10 |
| param_n_estimators | | | | |
| 5 | 0.954348 | 0.963043 | 0.836957 | 0.939130 |
| 10 | 0.847101 | 0.963043 | 0.952899 | 0.908696 |
| 50 | 0.953623 | 0.963043 | 0.963043 | 0.956522 |
| 150 | 0.953623 | 0.963043 | 0.963043 | 0.955072 |
| 200 | 0.952174 | 0.963043 | 0.962319 | 0.952899 |

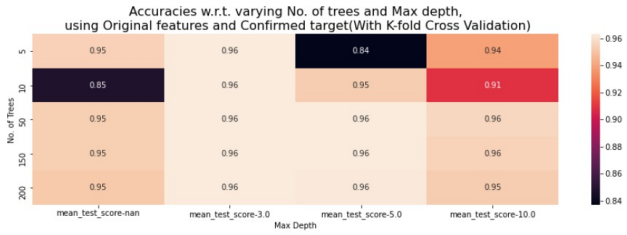
```
accuracies of varying No. of trees and Max depth, using Original features and
Deaths target(With K-fold Cross Validation) are:
```

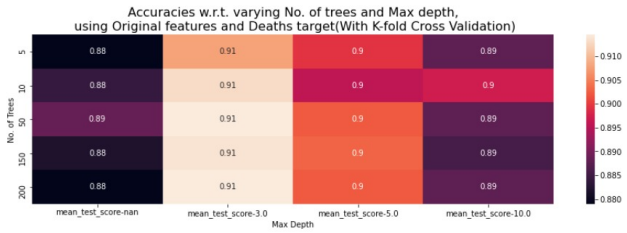
| | mean_test_score | | | |
|---|---|---|---|---|
| param_max_depth | NaN | 3 | 5 | 10 |
| param_n_estimators | | | | |
| 5 | 0.878986 | 0.907971 | 0.899275 | 0.887681 |
| 10 | 0.884058 | 0.913043 | 0.895652 | 0.897101 |
| 50 | 0.888406 | 0.914493 | 0.902174 | 0.887681 |
| 150 | 0.881884 | 0.913768 | 0.902899 | 0.885507 |
| 200 | 0.878986 | 0.914493 | 0.902174 | 0.887681 |

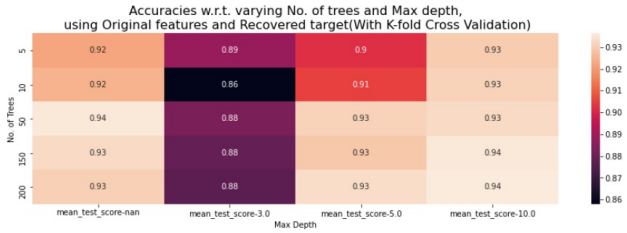
accuracies of varying No. of trees and Max depth, using Original features and Recovered target(With K-fold Cross Validation) are:

| | mean_test_score | | | |
|---|---|---|---|---|
| param_max_depth | NaN | 3 | 5 | 10 |
| param_n_estimators | | | | |
| 5 | 0.922464 | 0.887681 | 0.902174 | 0.929710 |
| 10 | 0.924638 | 0.857971 | 0.905072 | 0.931159 |
| 50 | 0.935507 | 0.882609 | 0.931159 | 0.932609 |
| 150 | 0.932609 | 0.878261 | 0.928986 | 0.935507 |
| 200 | 0.930435 | 0.876812 | 0.933333 | 0.936232 |





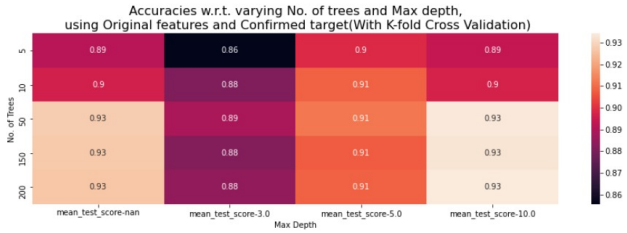



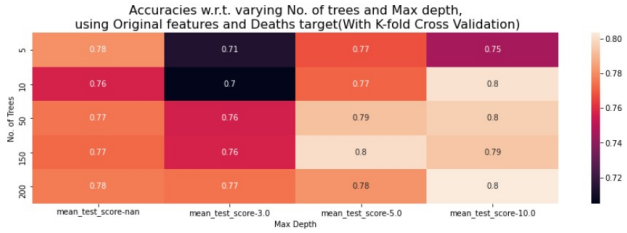

#### 0.8.5 With SMOTE:

[99]:
```
rfc_With_Validation_Original_Confirmed_SMOTE=rfc_With_validation(x_Confirmed,y_sm_Confirmed,'C
rfc_With_Validation_Original_Deaths_SMOTE=rfc_With_validation(x_Deaths,y_sm_Deaths,'Original',
```

```
accuracies of varying No. of trees and Max depth, using Original features and
Confirmed target(With K-fold Cross Validation) are:

                   mean_test_score
param_max_depth                NaN         3         5         10
param_n_estimators
5                         0.891234  0.855519  0.898799  0.893856
10                        0.896491  0.880729  0.908188  0.897240
50                        0.927752  0.888629  0.911948  0.933774
150                       0.927376  0.881852  0.907424  0.932640
200                       0.926623  0.884114  0.908179  0.934148

accuracies of varying No. of trees and Max depth, using Original features and
Deaths target(With K-fold Cross Validation) are:
```

| | mean_test_score | | | |
|---|---|---|---|---|
| param_max_depth | NaN | 3 | 5 | 10 |
| param_n_estimators | | | | |
| 5 | 0.780897 | 0.713745 | 0.767619 | 0.745551 |
| 10 | 0.757199 | 0.704916 | 0.771214 | 0.801026 |
| 50 | 0.774868 | 0.756421 | 0.792550 | 0.795763 |
| 150 | 0.772452 | 0.757209 | 0.799796 | 0.793742 |
| 200 | 0.776463 | 0.774511 | 0.781693 | 0.803399 |



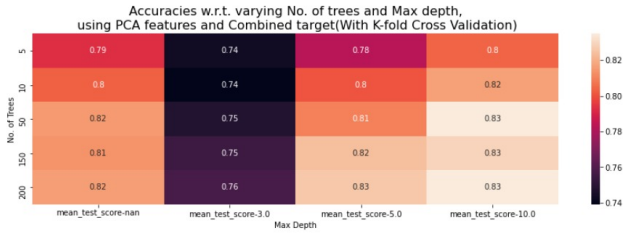




#### 0.8.6 Hyper Parameter tuning of No. of trees and max_depth usinf PCA features(With Validation):

[100]:
```
# Retreiving PCA data for Random Forest Classifier
pca_rfc=pca(x_train,9)

# Random Forest Classifier using PCA data
rfc_With_Validation_PCA=rfc_With_validation(pca_rfc,y_train,'PCA','Combined')
```

accuracies of varying No. of trees and Max depth, using PCA features and Combined target(With K-fold Cross Validation) are:

| | mean_test_score | | | |
|---|---|---|---|---|
| param_max_depth | NaN | 3 | 5 | 10 |
| param_n_estimators | | | | |
| 5 | 0.792754 | 0.743478 | 0.782609 | 0.803623 |
| 10 | 0.802899 | 0.739130 | 0.800725 | 0.816667 |
| 50 | 0.815217 | 0.747826 | 0.809420 | 0.834783 |
| 150 | 0.813768 | 0.754348 | 0.823913 | 0.828986 |
| 200 | 0.815942 | 0.757246 | 0.826087 | 0.834783 |




### 0.8.7 Hyperparameter tuning without Validation:

[101]:
```
# Function that returns RandomForestClassifier accuracy without validation
def rfc_Without_validation(x,y,str1,str2):
    rfc_scores=np.zeros((5,4))
    nEstimators=[5,10,50,150,200]
    maxDepth=[3,5,10,None]
    for i in range(len(nEstimators)):
        for j in range(len(maxDepth)):
            rfc1= RandomForestClassifier(n_estimators=nEstimators[i],␣
↪max_depth=maxDepth[j]).fit(x_train,y_train)
            rfc_scores[i][j]= (accuracy_score(y_train,rfc1.predict(x_train)))
    # Creating dataframe of accuracies
    rfc_scores=pd.
↪DataFrame(rfc_scores,columns=["Max_depth=3","Max_depth=5","Max_depth=10","Max_depth=None"])
    rfc_scores['No. of trees']=nEstimators
    rfc_scores.set_index('No. of trees',inplace=True)

    # Plotting Heatmap of accuracies achieved without validation
    plt.figure(figsize=(15,4))
    sns.heatmap(rfc_scores,annot=True)
```

```
    plt.title("Accuracies w.r.t. varying No. of trees and Max depth, using {}␣
↪features and {} target(w/o validation)".format(str1,str2),fontsize=16)
    # Printing exact accuracies
    print('accuracies of varying No. of trees and Max depth, using {} features␣
↪and {} target(w/o validation) are: '.format(str1,str2))
    display(rfc_scores)
```

#### 0.8.8 Hyperparameter tuning on original features(Without Validation):

#### 0.8.9 Without SMOTE:

[102]:
```
rfc_Without_Validation_Original_Combined=rfc_Without_validation(x_train,y_train,'Original','Co
rfc_Without_Validation_Original_Confirmed=rfc_Without_validation(x_train,y_train['Confirmed'],
rfc_Without_Validation_Original_Deaths=rfc_Without_validation(x_train,y_train['Deaths'],'Origi
rfc_Without_Validation_Original_Recovered=rfc_Without_validation(x_train,y_train['Recovered'],
```

```
accuracies of varying No. of trees and Max depth, using Original features and
Combined target(w/o validation) are:
```

| No. of trees | Max_depth=3 | Max_depth=5 | Max_depth=10 | Max_depth=None |
|---|---|---|---|---|
| 5 | 0.833333 | 0.818841 | 0.917391 | 0.973188 |
| 10 | 0.794928 | 0.856522 | 0.932609 | 0.989855 |
| 50 | 0.781884 | 0.850000 | 0.924638 | 0.998551 |
| 150 | 0.796377 | 0.852899 | 0.922464 | 0.999275 |
| 200 | 0.787681 | 0.856522 | 0.926087 | 1.000000 |

```
accuracies of varying No. of trees and Max depth, using Original features and
Confirmed target(w/o validation) are:
```

| No. of trees | Max_depth=3 | Max_depth=5 | Max_depth=10 | Max_depth=None |
|---|---|---|---|---|
| 5 | 0.786232 | 0.818116 | 0.913768 | 0.971739 |
| 10 | 0.761594 | 0.856522 | 0.926812 | 0.985507 |
| 50 | 0.823913 | 0.852899 | 0.921014 | 0.998551 |
| 150 | 0.797826 | 0.855072 | 0.923913 | 1.000000 |
| 200 | 0.762319 | 0.856522 | 0.921014 | 1.000000 |

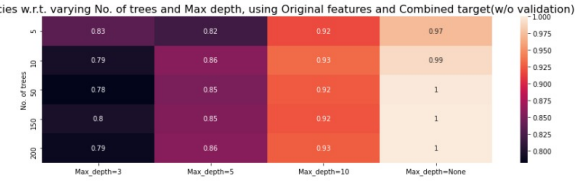
```
accuracies of varying No. of trees and Max depth, using Original features and
Deaths target(w/o validation) are:
```

| No. of trees | Max_depth=3 | Max_depth=5 | Max_depth=10 | Max_depth=None |
|---|---|---|---|---|
| 5 | 0.753623 | 0.854348 | 0.913768 | 0.968841 |
| 10 | 0.755797 | 0.833333 | 0.936232 | 0.981884 |
| 50 | 0.761594 | 0.844928 | 0.926812 | 0.998551 |
| 150 | 0.813768 | 0.852899 | 0.922464 | 1.000000 |
| 200 | 0.796377 | 0.855072 | 0.928986 | 1.000000 |

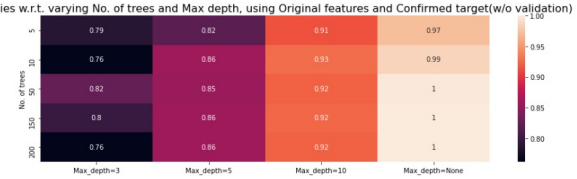
```
accuracies of varying No. of trees and Max depth, using Original features and
Recovered target(w/o validation) are:
```

| No. of trees | Max_depth=3 | Max_depth=5 | Max_depth=10 | Max_depth=None |
|---|---|---|---|---|
| 5 | 0.786232 | 0.864493 | 0.918116 | 0.970290 |
| 10 | 0.795652 | 0.869565 | 0.922464 | 0.982609 |
| 50 | 0.814493 | 0.855797 | 0.927536 | 0.999275 |
| 150 | 0.794928 | 0.852899 | 0.921014 | 1.000000 |
| 200 | 0.794203 | 0.853623 | 0.926087 | 1.000000 |

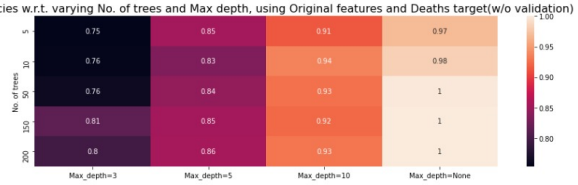
Accuracies w.r.t. varying No. of trees and Max depth, using Original features and Combined target(w/o validation)

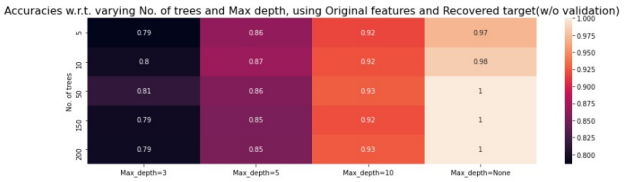



Accuracies w.r.t. varying No. of trees and Max depth, using Original features and Confirmed target(w/o validation)




Accuracies w.r.t. varying No. of trees and Max depth, using Original features and Deaths target(w/o validation)






### 0.8.10 With SMOTE:

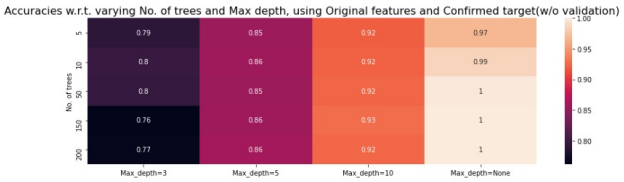
[103]:
```
rfc_Without_Validation_Original_Confirmed_SMOTE=rfc_Without_validation(x_Confirmed,y_sm_Confir
rfc_Without_Validation_Original_Deaths_SMOTE=rfc_Without_validation(x_Deaths,y_sm_Deaths,'Orig
```

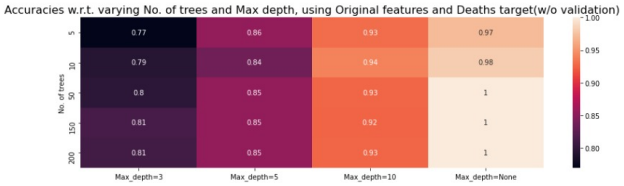
```
accuracies of varying No. of trees and Max depth, using Original features and
Confirmed target(w/o validation) are:
```

| No. of trees | Max_depth=3 | Max_depth=5 | Max_depth=10 | Max_depth=None |
|---|---|---|---|---|
| 5 | 0.789855 | 0.854348 | 0.919565 | 0.968116 |
| 10 | 0.797101 | 0.855797 | 0.921014 | 0.985507 |
| 50 | 0.795652 | 0.854348 | 0.923188 | 0.998551 |
| 150 | 0.761594 | 0.855072 | 0.926812 | 1.000000 |
| 200 | 0.765217 | 0.857971 | 0.923913 | 1.000000 |

```
accuracies of varying No. of trees and Max depth, using Original features and
Deaths target(w/o validation) are:
```

| No. of trees | Max_depth=3 | Max_depth=5 | Max_depth=10 | Max_depth=None |
|---|---|---|---|---|
| 5 | 0.769565 | 0.855072 | 0.928986 | 0.971739 |
| 10 | 0.793478 | 0.835507 | 0.940580 | 0.981884 |
| 50 | 0.797101 | 0.854348 | 0.925362 | 0.999275 |
| 150 | 0.813043 | 0.854348 | 0.924638 | 1.000000 |
| 200 | 0.810870 | 0.854348 | 0.925362 | 1.000000 |



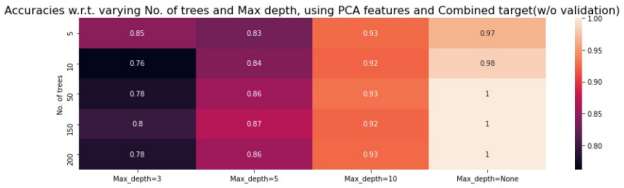



### 0.8.11 Hyperparameter tuning on PCA features(Without Validation):

[104]:
```
rfc_Without_Validation_PCA=rfc_Without_validation(pca_rfc,y_train,'PCA','Combined')
```

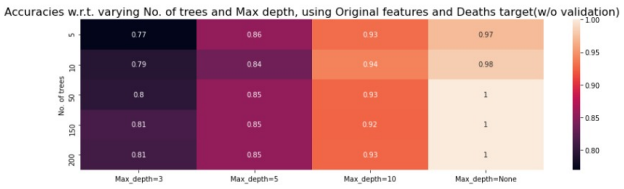
```
accuracies of varying No. of trees and Max depth, using PCA features and
Combined target(w/o validation) are:
```

| No. of trees | Max_depth=3 | Max_depth=5 | Max_depth=10 | Max_depth=None |
|---|---|---|---|---|
| 5 | 0.845652 | 0.831884 | 0.926087 | 0.965942 |
| 10 | 0.761594 | 0.841304 | 0.922464 | 0.984058 |
| 50 | 0.777536 | 0.855797 | 0.929710 | 1.000000 |
| 150 | 0.797101 | 0.869565 | 0.923188 | 1.000000 |
| 200 | 0.781159 | 0.855072 | 0.925362 | 1.000000 |




From PCA features, it is observed that the maximum accuracy is achieved at no. of trees = 150 and 200 and max depth = 5 and 10, but common among both maximum accuracy is achievedat no. of trees =150 and max_depth= 10. Also, looking individually at each feature, they have both have good results among their range.

## 0.9 CM[5]:

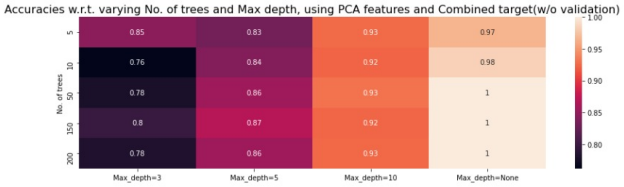



### 0.8.11 Hyperparameter tuning on PCA features(Without Validation):

[104]:
```
rfc_Without_Validation_PCA=rfc_Without_validation(pca_rfc,y_train,'PCA','Combined')
```

```
accuracies of varying No. of trees and Max depth, using PCA features and
Combined target(w/o validation) are:
```

| No. of trees | Max_depth=3 | Max_depth=5 | Max_depth=10 | Max_depth=None |
|---|---|---|---|---|
| 5 | 0.845652 | 0.831884 | 0.926087 | 0.965942 |
| 10 | 0.761594 | 0.841304 | 0.922464 | 0.984058 |
| 50 | 0.777536 | 0.855797 | 0.929710 | 1.000000 |
| 150 | 0.797101 | 0.869565 | 0.923188 | 1.000000 |
| 200 | 0.781159 | 0.855072 | 0.925362 | 1.000000 |




From PCA features, it is observed that the maximum accuracy is achieved at no. of trees = 150 and 200 and max depth = 5 and 10, but common among both maximum accuracy is achievedat no. of trees =150 and max_depth= 10. Also, looking individually at each feature, they have both have good results among their range.

## 0.9 CM[5]:

#### 0.9.1 Gradient Boosting Classifier:

1) With Validation:
   - Without SMOTE
   a) Best n_estimators for Combined targets is 50
   b) Best n_estimators for 'Confirmed' targets is 5
   c) Best n_estimators for 'Deaths' target is 10
   d) Best n_estimators for 'Recovered' target is 50
   - With SMOTE
   a) Best n_estimators for 'Confirmed' targets is 150
   b) Best n_estimators for 'Deaths' target is 5
2) Without Validation:
   - Without SMOTE
   a) Best n_estimators for Combined targets is 200
   b) Best n_estimators for 'Confirmed' targets is 200
   c) Best n_estimators for 'Deaths' target is 200
   d) Best n_estimators for 'Recovered' target is 200
   - With SMOTE
   a) Best n_estimators for 'Confirmed' targets is 200
   b) Best n_estimators for 'Deaths' target is 200
3) PCA Features:
   a)With Validation: Best n_estimators for targets is 50
   b)Without Validation: Best n_estimators for targets is 200

So, n_estimators here treats every target differently and , in maximum cases(both original and PCA) n_estimators=50 performed better in validation whereas, without validation 200 no. of trees performs better. Although, their combination may provide better results.

#### 0.9.2 Hyperparameter tuning with K-fold Cross Validation:

[105]:
```
# Function that returns GradientBoostingClassifier accuracy with kfold cross␣
↪validation
def gbc_With_validation(x,y,str1,str2):
    # GradientBoostingClassifier using kfold cross validation on original data
    # setting hyperparameters
    nEstimators=[5,10,50,150,200]
    gbc_df=[]
    for i in nEstimators:
```

```
        if str2=='Combined':
            # applying multioutput on GradientBoostingClassifier, as there are multiple targets
            gbc=multioutput.MultiOutputClassifier(GradientBoostingClassifier(n_estimators=i))
        else:
            gbc=GradientBoostingClassifier(n_estimators=i)
        gbc_df.append((cross_val_score(gbc,x,y,cv=5)).mean())

    # Converting list to dataframe
    gbc_df=pd.DataFrame(gbc_df,columns=['Mean Accuracy rate'])
    gbc_df['n_estimators']=nEstimators
    gbc_df.set_index('n_estimators',inplace=True)

    # Plotting graph of No. of Trees vs. Accuracy Rate on original features
    plt.figure(figsize=(15,3))
    plt.plot(nEstimators,gbc_df['Mean Accuracy rate'],marker='o',markerfacecolor='r',color='g')
    plt.xlabel("No. of estimators")
    plt.ylabel(" Mean Accuracy Rate")
    plt.title("No. of estimators vs. Accuracy Rate on {} features and {} target(With Cross Validation)".format(str1,str2), fontsize=16, fontweight='bold')
    plt.grid()
    # Printing exact values
    print("Accuracies w.r.t. varying No. of estimators, using  {} features and {} target(With Cross Validation): ".format(str1,str2))
    display(gbc_df)
```

#### 0.9.3 Hyper Parameter tuning(n_estimators), using Original features(With Validation):

#### 0.9.4 Without SMOTE:

[106]:
```
gbc_With_CV_Original_Combined=gbc_With_validation(x_train,y_train,'Original','Combined')
gbc_With_CV_Original_Confirmed=gbc_With_validation(x_train,y_train['Confirmed'],'Original','Co
gbc_With_CV_Original_Deaths=gbc_With_validation(x_train,y_train['Deaths'],'Original','Deaths')
gbc_With_CV_Original_Recovered=gbc_With_validation(x_train,y_train['Recovered'],'Original','Re
```

```
Accuracies w.r.t. varying No. of estimators, using  Original features and
Combined target(With Cross Validation):
```

| n_estimators | Mean Accuracy rate |
|---|---|
| 5 | 0.748551 |
| 10 | 0.797826 |
| 50 | 0.840580 |
| 150 | 0.833333 |

```
200                     0.826087
```

```
Accuracies w.r.t. varying No. of estimators, using  Original features and
Confirmed target(With Cross Validation):
```

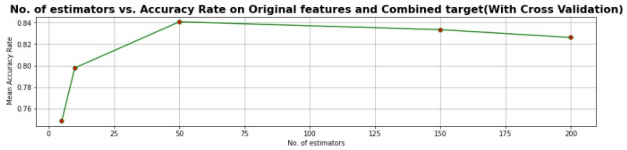
```
              Mean Accuracy rate
n_estimators
5                       0.936957
10                      0.817391
50                      0.813768
150                     0.845652
200                     0.823913
```

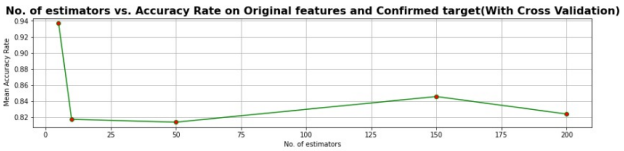
```
Accuracies w.r.t. varying No. of estimators, using  Original features and Deaths
target(With Cross Validation):
```

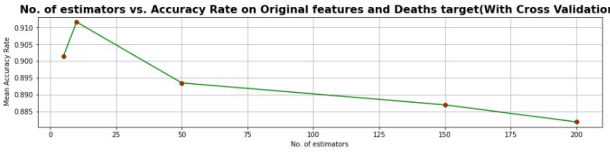
```
              Mean Accuracy rate
n_estimators
5                       0.901449
10                      0.911594
50                      0.893478
150                     0.886957
200                     0.881884
```

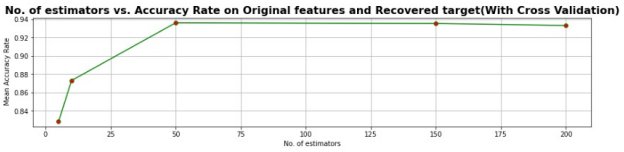
```
Accuracies w.r.t. varying No. of estimators, using  Original features and
Recovered target(With Cross Validation):
```

```
              Mean Accuracy rate
n_estimators
5                       0.828261
10                      0.873188
50                      0.936232
150                     0.935507
200                     0.933333
```





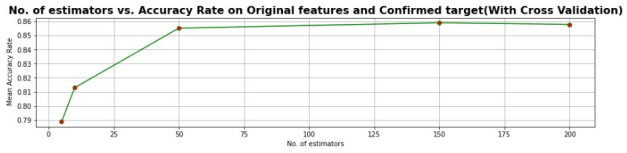

#### 0.9.5 With SMOTE:

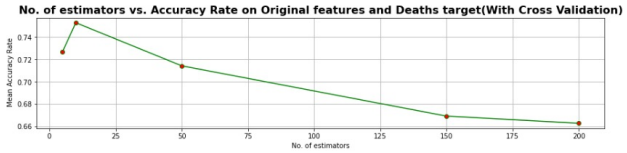
[107]:
```
gbc_With_CV_Original_Confirmed_SMOTE=gbc_With_validation(x_Confirmed,y_sm_Confirmed,'Original'
gbc_With_CV_Original_Deaths_SMOTE=gbc_With_validation(x_Deaths,y_sm_Deaths,'Original','Deaths'
```

```
Accuracies w.r.t. varying No. of estimators, using  Original features and
Confirmed target(With Cross Validation):
```

| n_estimators | Mean Accuracy rate |
|---|---|
| 5 | 0.788858 |
| 10 | 0.812929 |
| 50 | 0.855075 |
| 150 | 0.858840 |
| 200 | 0.857713 |

```
Accuracies w.r.t. varying No. of estimators, using  Original features and Deaths
target(With Cross Validation):

              Mean Accuracy rate
n_estimators
5                       0.726996
10                      0.752745
50                      0.714177
150                     0.669184
200                     0.662763
```



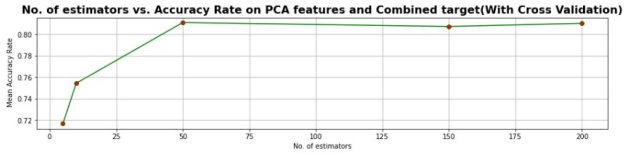




#### 0.9.6 Hyper Parameter tuning( n_estimators), using PCA features(With Validation):

[108]:
```
# Retreiving PCA data for PCA data
pca_gbc= pca(x_train,9)

# GradientBoostingClassifier using kfold cross validation on PCA data
gbc_With_CV_PCA=gbc_With_validation(pca_gbc,y_train,'PCA','Combined')
```

```
Accuracies w.r.t. varying No. of estimators, using  PCA features and Combined
target(With Cross Validation):

              Mean Accuracy rate
n_estimators
5                       0.716667
10                      0.754348
50                      0.810870
```

```
150                    0.807246
200                    0.810145
```




#### 0.9.7 Hyperparameter tuning without Validation:

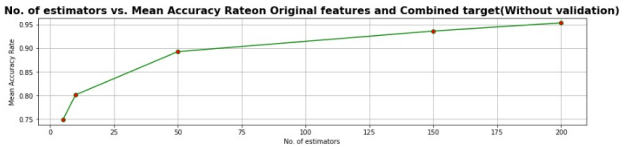
[109]:
```
# Function that returns GradientBoostingClassifier accuracy without validation
def gbc_Without_validation(x,y,str1,str2):
    # accuracies using training set without validation
    scores=[]
    nEstimators=[5,10,50,150,200]
    for i in nEstimators:
        if str2=='Combined':
            gbc=multioutput.
↪MultiOutputClassifier(GradientBoostingClassifier(n_estimators=i)).fit(x,y)
        else:
            gbc=GradientBoostingClassifier(n_estimators=i).fit(x,y)
        scores.append(accuracy_score(y,gbc.predict(x)))
    # Plotting graph of Max Depth vs. Accuracy Rate without validation
    plt.figure(figsize=(15,3))
    plt.plot(nEstimators,scores,marker='o',markerfacecolor='r',color='g')
    plt.xlabel("No. of estimators")
    plt.ylabel("Mean Accuracy Rate")
    plt.title("No. of estimators vs. Mean Accuracy Rateon {} features and {}␣
↪target(Without validation)".format(str1,str2), fontsize=16,
              fontweight='bold')
    plt.grid()

    # Converting list to dataframe
    scores=pd.DataFrame(scores,columns=['Training_Set_Accuracy'])
    scores['n_estimators']=nEstimators
    scores.set_index('n_estimators',inplace=True)

    # Printing exact accuracies
    print('The accuracy scores on {} features and {} target(Without validation)␣
↪are:'.format(str1,str2))
    display(scores)
```

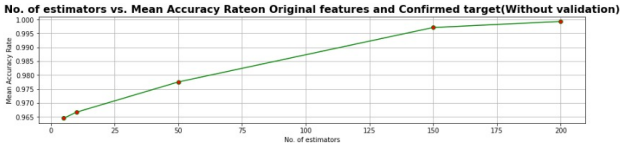
#### 0.9.8 Hyperparameter tuning on original features(Without validation):

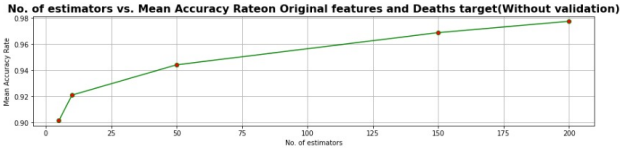
#### 0.9.9 Without SMOTE:

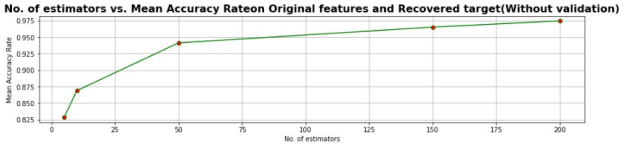
[110]:
```
gbc_Without_CV_Original_Combined=gbc_Without_validation(x_train,y_train,'Original','Combined')
gbc_Without_CV_Original_Confirmed=gbc_Without_validation(x_train,y_train['Confirmed'],'Origina
gbc_Without_CV_Original_Deaths=gbc_Without_validation(x_train,y_train['Deaths'],'Original','De
gbc_Without_CV_Original_Recovered=gbc_Without_validation(x_train,y_train['Recovered'],'Origina
```

```
The accuracy scores on Original features and Combined target(Without validation)
are:

              Training_Set_Accuracy
n_estimators
5                          0.748551
10                         0.801449
50                         0.892754
150                        0.936232
200                        0.953623

The accuracy scores on Original features and Confirmed target(Without
validation) are:

              Training_Set_Accuracy
n_estimators
5                          0.964493
10                         0.966667
50                         0.977536
150                        0.997101
200                        0.999275

The accuracy scores on Original features and Deaths target(Without validation)
are:

              Training_Set_Accuracy
n_estimators
5                          0.901449
10                         0.921014
50                         0.944203
150                        0.968841
200                        0.977536

The accuracy scores on Original features and Recovered target(Without
validation) are:

              Training_Set_Accuracy
n_estimators
5                          0.828261
10                         0.868841
50                         0.941304
150                        0.965217
200                        0.974638
```




#### 0.9.10 With SMOTE:

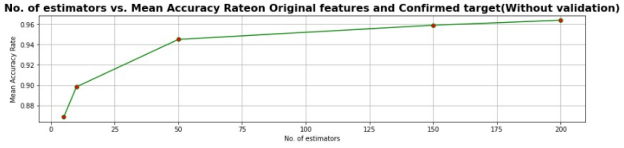
[111]:
```
gbc_Without_CV_Original_Confirmed_SMOTE=gbc_Without_validation(x_Confirmed,y_sm_Confirmed,'Ori
gbc_Without_CV_Original_Deaths_SMOTE=gbc_Without_validation(x_Deaths,y_sm_Deaths,'Original','I
```

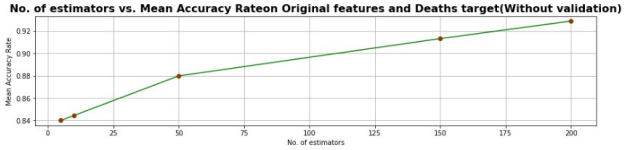
The accuracy scores on Original features and Confirmed target(Without validation) are:

| n_estimators | Training_Set_Accuracy |
|---|---|
| 5 | 0.868698 |
| 10 | 0.898420 |
| 50 | 0.945071 |
| 150 | 0.958992 |
| 200 | 0.963883 |

The accuracy scores on Original features and Deaths target(Without validation) are:

| n_estimators | Training_Set_Accuracy |
|---|---|
| 5 | 0.840032 |
| 10 | 0.844453 |
| 50 | 0.879823 |
| 150 | 0.913183 |
| 200 | 0.928859 |



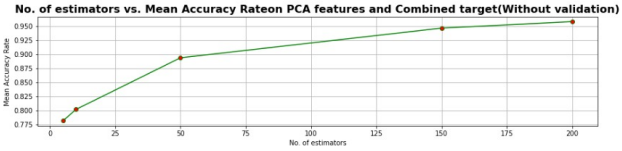




### 0.9.11 Hyperparameter tuning on PCA features(Without validation):

[112]:
```
gbc_Without_Validation_PCA=␣
↪gbc_Without_validation(pca_gbc,y_train,'PCA','Combined')
```

The accuracy scores on PCA features and Combined target(Without validation) are:

| n_estimators | Training_Set_Accuracy |
|---|---|
| 5 | 0.781159 |
| 10 | 0.801449 |
| 50 | 0.893478 |
| 150 | 0.946377 |
| 200 | 0.957971 |




### 0.10 CM[6]:

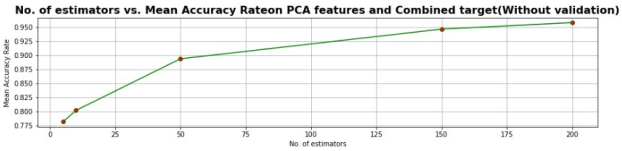
#### 0.10.1 Naive Bayes Classifier

1) With Validation:
   - Without SMOTE
   a) Best var_smoothing for Combined targets is 1e-1
   b) Best var_smoothing for 'Confirmed' targets is 1e-10
   c) Best var_smoothing for 'Deaths' target is 1e-1
   d) Best var_smoothing for 'Recovered' target is 1e-10
   - With SMOTE
   a) Best var_smoothing for 'Confirmed' targets is 1e-10
   b) Best var_smoothing for 'Deaths' target is 1e-10
2) Without Validation:
   - Without SMOTE
   a) Best var_smoothing for Combined targets is 1e-1
   b) Best var_smoothing for 'Confirmed' targets is 1e-1
   c) Best var_smoothing for 'Deaths' target is 1e-1
   d) Best var_smoothing for 'Recovered' target is 1e-10
   - With SMOTE
   a) Best var_smoothing for 'Confirmed' targets is 1e-10
   b) Best var_smoothing for 'Deaths' target is 1e-10

So, var_smoothing here treats every target differently and ,as recovered feature is balanced target in first thus proved better results at 1e-10 in both cases. Also, when we balance other features with SMOTE var_smoothing provides better accuracies at 1e-10. So, it can be said that for balanced data , max accuracy is achieved at 1e-10 and for imabalanced target or combination of targets maximum accuracy is achieved at 1e-1.

-Thus, here smoothing impacts at the type of data, in imbalanced , it performed well with less value

| n_estimators | Training_Set_Accuracy |
|---|---|
| 5 | 0.781159 |
| 10 | 0.801449 |
| 50 | 0.893478 |
| 150 | 0.946377 |
| 200 | 0.957971 |




### 0.10 CM[6]:

#### 0.10.1 Naive Bayes Classifier

1) With Validation:
   - Without SMOTE
   a) Best var_smoothing for Combined targets is 1e-1
   b) Best var_smoothing for 'Confirmed' targets is 1e-10
   c) Best var_smoothing for 'Deaths' target is 1e-1
   d) Best var_smoothing for 'Recovered' target is 1e-10
   - With SMOTE
   a) Best var_smoothing for 'Confirmed' targets is 1e-10
   b) Best var_smoothing for 'Deaths' target is 1e-10
2) Without Validation:
   - Without SMOTE
   a) Best var_smoothing for Combined targets is 1e-1
   b) Best var_smoothing for 'Confirmed' targets is 1e-1
   c) Best var_smoothing for 'Deaths' target is 1e-1
   d) Best var_smoothing for 'Recovered' target is 1e-10
   - With SMOTE
   a) Best var_smoothing for 'Confirmed' targets is 1e-10
   b) Best var_smoothing for 'Deaths' target is 1e-10

So, var_smoothing here treats every target differently and ,as recovered feature is balanced target in first thus proved better results at 1e-10 in both cases. Also, when we balance other features with SMOTE var_smoothing provides better accuracies at 1e-10. So, it can be said that for balanced data , max accuracy is achieved at 1e-10 and for imabalanced target or combination of targets maximum accuracy is achieved at 1e-1.

-Thus, here smoothing impacts at the type of data, in imbalanced , it performed well with large

value of var_smoothing and for balanced data it performed well with less value of var_smoothing. So, it can be said that it performs different with imbalanced data.Thus, as its value increases, it is more suitable for imbalanced data and less suitable for balanced data. Also, it ensures that all feature values have non-zero probability. Therefore, more smoothing is required for imbalanced data and less for balanced data.

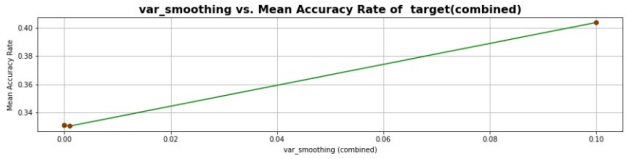
#### 0.10.2 Hyperparameter tuning with 10-Fold Cross Validation:

[113]:
```
# Function that returns GaussianNB Classifier accuracy with kfold cross␣
↪validation
def gaussianNB_Validation(x,y,output_type,col,str1):
    # setting hyperparameters
    varSmoothing=[1e-10,1e-9,1e-5,1e-3,1e-1]
    nbc_df=[]
    # for multitarget outputs
    if output_type== 'combined':
        for i in varSmoothing:
            # applying multioutput on GaussianNB Classifier, as there are␣
↪multiple targets
            nbc=multioutput.MultiOutputClassifier(GaussianNB(var_smoothing=i))
            nbc_df.append((cross_val_score(nbc,x,y,cv=10)).mean())
    # for individual outputs
    elif output_type=='individually':
        if col!= None:
            y=y[col]
        for i in varSmoothing:
            # applying GaussianNB Classifier, individually
            nbc=GaussianNB(var_smoothing=i)
            nbc_df.append((cross_val_score(nbc,x,y,cv=10)).mean())
    # Converting list to dataframe
    nbc_scores=pd.DataFrame(nbc_df,columns=['Mean Accuracy rate'])
    nbc_scores['var_smoothing']=varSmoothing
    nbc_scores.set_index('var_smoothing',inplace=True)

    # Plotting graph of No. of Trees vs. Accuracy Rate on original features
    plt.figure(figsize=(15,3))
    plt.plot(varSmoothing,nbc_df,marker='o',markerfacecolor='r',color='g')
    plt.xlabel("var_smoothing ({})".format(output_type))
    plt.ylabel("Mean Accuracy Rate")
    plt.title("var_smoothing vs. Mean Accuracy Rate of {} target({})".
↪format(str1,output_type), fontsize=16, fontweight='bold')
    plt.grid()

    # Printing exact values
    print("Accuracies w.r.t. varying var_smoothing of {} target({}): ".
↪format(str1,output_type))
    display(nbc_scores)
```

#### 0.10.3 HyperParameter tuning of var_smoothing, using Original features with 10-Fold Cross Validation(Combined):

[114]:
```
nbc_Combined_With_Validation=gaussianNB_Validation(x_train,y_train,'combined',None,'')
```

```
Accuracies w.r.t. varying var_smoothing of  target(combined):

               Mean Accuracy rate
var_smoothing
1.000000e-10             0.331159
1.000000e-09             0.331159
1.000000e-05             0.331159
1.000000e-03             0.330435
1.000000e-01             0.403623
```




#### 0.10.4 Hyper Parameter tuning of var_smoothing, using Original features with 10-Fold Cross Validation(Individually):

#### 0.10.5 Without SMOTE:

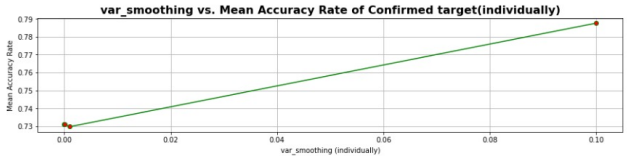
[115]:
```
nbc_Confirmed_With_Validation=gaussianNB_Validation(x_train,y_train,'individually','Confirmed'
nbc_Deaths_With_Validation=gaussianNB_Validation(x_train,y_train,'individually','Deaths','Deat
nbc_Recovered_With_Validation=gaussianNB_Validation(x_train,y_train,'individually','Recovered'
```

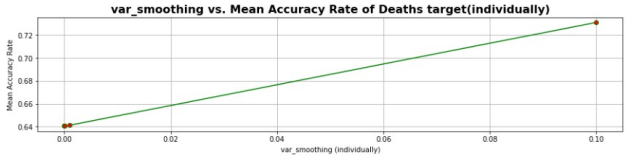
```
Accuracies w.r.t. varying var_smoothing of Confirmed target(individually):

               Mean Accuracy rate
var_smoothing
1.000000e-10             0.731159
1.000000e-09             0.731159
1.000000e-05             0.731159
1.000000e-03             0.729710
1.000000e-01             0.787681

Accuracies w.r.t. varying var_smoothing of Deaths target(individually):

               Mean Accuracy rate
var_smoothing
1.000000e-10             0.640580
1.000000e-09             0.640580
```

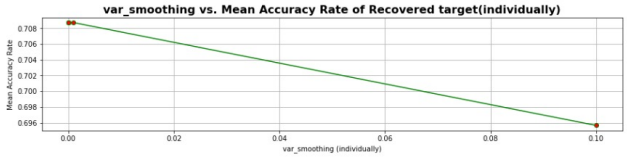
```
1.000000e-05              0.640580
1.000000e-03              0.641304
1.000000e-01              0.731159
```

Accuracies w.r.t. varying var_smoothing of Recovered target(individually):

```
               Mean Accuracy rate
var_smoothing
1.000000e-10              0.708696
1.000000e-09              0.708696
1.000000e-05              0.708696
1.000000e-03              0.708696
1.000000e-01              0.695652
```



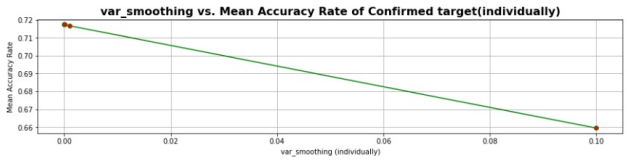



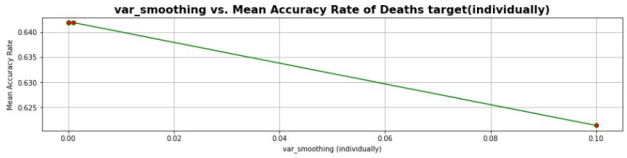



#### 0.10.6 With SMOTE:

[116]:
```
nbc_Confirmed_With_Validation_SMOTE=gaussianNB_Validation(x_Confirmed,y_sm_Confirmed,'individu
nbc_Deaths_With_Validation_SMOTE=gaussianNB_Validation(x_Deaths,y_sm_Deaths,'individually',Nor
```

```
Accuracies w.r.t. varying var_smoothing of Confirmed target(individually):

               Mean Accuracy rate
var_smoothing
1.000000e-10             0.717458
1.000000e-09             0.717458
1.000000e-05             0.717458
1.000000e-03             0.716706
1.000000e-01             0.659516

Accuracies w.r.t. varying var_smoothing of Deaths target(individually):

               Mean Accuracy rate
var_smoothing
1.000000e-10             0.641879
1.000000e-09             0.641879
1.000000e-05             0.641879
1.000000e-03             0.641879
1.000000e-01             0.621382
```



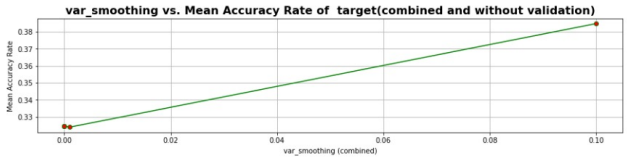
#### 0.10.7 Hyperparameter tuning without Validation:

[117]:
```
# Function that returns GaussianNB Classifier accuracy without validation
def gaussianNB_Without_Validation(x,y,output_type,col,str1):
    # setting hyperparameters
    varSmoothing=[1e-10,1e-9,1e-5,1e-3,1e-1]
    nbc_df=[]
    # for multitarget outputs
    if(output_type== 'combined'):
        for i in varSmoothing:
            # applying multioutput on GaussianNB Classifier, as there are␣
↪multiple targets
            nbc=multioutput.MultiOutputClassifier(GaussianNB(var_smoothing=i)).
↪fit(x,y)
            nbc_df.append(accuracy_score(y,nbc.predict(x)))
    # for individual outputs
    elif(output_type=='individually'):
        if col==None:
            y=y
        else:
            y=y[col]
        for i in varSmoothing:
            # applying GaussianNB Classifier, individually
            nbc=GaussianNB(var_smoothing=i).fit(x,y)
            nbc_df.append(accuracy_score(y,nbc.predict(x)))
    # Converting list to dataframe
    nbc_scores=pd.DataFrame(nbc_df,columns=['Mean Accuracy rate'])
    nbc_scores['var_smoothing']=varSmoothing
    nbc_scores.set_index('var_smoothing',inplace=True)

    # Plotting graph of No. of Trees vs. Accuracy Rate on original features
    plt.figure(figsize=(15,3))
    plt.plot(varSmoothing,nbc_df,marker='o',markerfacecolor='r',color='g')
    plt.xlabel("var_smoothing ({})".format(output_type))
    plt.ylabel("Mean Accuracy Rate")
    plt.title("var_smoothing vs. Mean Accuracy Rate of {} target({} and without␣
↪validation)".format(str1,output_type), fontsize=16, fontweight='bold')
    plt.grid()

    # Printing exact values
    print("Accuracies w.r.t. varying var_smoothing of {} target({} and without␣
↪validation): ".format(str1,output_type))
    display(nbc_scores)
```

#### 0.10.8 HyperParameter tuning of var_smoothing, using Original features without Validation(Combined):

[118]:
```
nbc_Combined_Without_Validation=gaussianNB_Without_Validation(x_train,y_train,'combined',None,
```

```
Accuracies w.r.t. varying var_smoothing of  target(combined and without
validation):

               Mean Accuracy rate
var_smoothing
1.000000e-10             0.324638
1.000000e-09             0.324638
1.000000e-05             0.324638
1.000000e-03             0.323913
1.000000e-01             0.384783
```




#### 0.10.9 Hyper Parameter tuning of var_smoothing, using Original features without Validation(Individually):

#### 0.10.10 Without SMOTE:

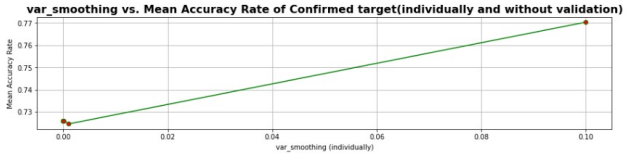
[119]:
```
nbc_Confirmed_Without_Validation=gaussianNB_Without_Validation(x_train,y_train,'individually',
nbc_Deaths_Without_Validation=gaussianNB_Without_Validation(x_train,y_train,'individually','De
nbc_Recovered_Without_Validation=gaussianNB_Without_Validation(x_train,y_train,'individually',
```

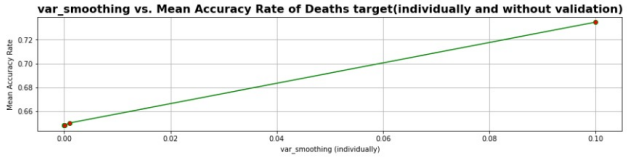
```
Accuracies w.r.t. varying var_smoothing of Confirmed target(individually and
without validation):

               Mean Accuracy rate
var_smoothing
1.000000e-10             0.726087
1.000000e-09             0.726087
1.000000e-05             0.726087
1.000000e-03             0.724638
1.000000e-01             0.770290

Accuracies w.r.t. varying var_smoothing of Deaths target(individually and
without validation):
```

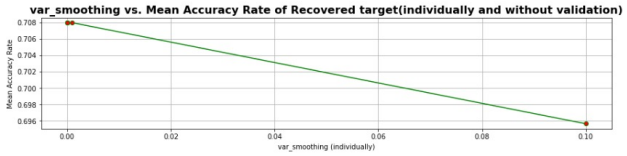
```
                Mean Accuracy rate
var_smoothing
1.000000e-10            0.647826
1.000000e-09            0.647826
1.000000e-05            0.647826
1.000000e-03            0.650000
1.000000e-01            0.734783
```

Accuracies w.r.t. varying var_smoothing of Recovered target(individually and without validation):

```
                Mean Accuracy rate
var_smoothing
1.000000e-10            0.707971
1.000000e-09            0.707971
1.000000e-05            0.707971
1.000000e-03            0.707971
1.000000e-01            0.695652
```



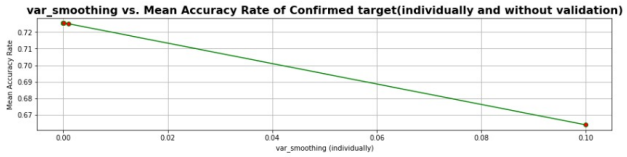



#### 0.10.11 With SMOTE:

[120]:
```
nbc_Confirmed_Without_Validation_SMOTE=gaussianNB_Without_Validation(x_Confirmed,y_sm_Confirme
nbc_Deaths_Without_Validation_SMOTE=gaussianNB_Without_Validation(x_Deaths,y_sm_Deaths,'indivi
```

```
Accuracies w.r.t. varying var_smoothing of Confirmed target(individually and
without validation):

               Mean Accuracy rate
var_smoothing
1.000000e-10             0.725357
1.000000e-09             0.725357
1.000000e-05             0.725357
1.000000e-03             0.724981
1.000000e-01             0.664033

Accuracies w.r.t. varying var_smoothing of Deaths target(individually and
without validation):

               Mean Accuracy rate
var_smoothing
1.000000e-10             0.647508
1.000000e-09             0.647508
1.000000e-05             0.647508
1.000000e-03             0.646704
1.000000e-01             0.627412
```




### 0.11 CM[7]:

#### 0.11.1 Interpretability:

Decision tree performs better than Naive bayes classifier even at non optimal parameter. Decision tree produced around 80% accuracy in imbalanced dataset whereas the maximum accuracy that NB achieved is around 40% . Also, if the data is balanced like in individual accuracy calculation of different targets, decision tree is able to get around 90% accuracy. On the other hand, even if the data is balanced NB achieved maximum upto 78% accuracy. Although, smoothing parameter helped in increasing accuracy but still less accurate than decision tree classifier. As, it has multiple targets and is supported by Decision tree classifier but, need to call another multiout classifier for Naive bayes. Even so, decision tree performed better.

And if we talk about without validation part, decision tree even reached upto 100% accuracy(although it may be because of over fitting but more, efficient for the data type provided). Decision tree handles imbalanced data better than Naive bayes.

1) So, decision tree may lead to over fitting whereas Naive bayes includes less over fitting.
2) Decision tree provides better accuracy , easy to understand than Naive bayes.
3) Naive bayes works well with small data whereas decision tree can handle large data as provided
4) Decision trees classifier is although slow as compared to naive bayes.

Also, Naive bayes is greatly effected with the type of data. In case of imbalanced data, it provided good accuracy at large var_smoothing of 1e-1 Whereas, in case of balanced data , it performed better at low var_smoothing value that is 1e-10 So, as compared to decision trees , NB is effected greatly with balanced/imbalanced data.

ECE 657A: Data and Knowledge Modelling and Analysis

Assignment 2: Classification using Naïve Bayes, Decision Tree, Random Forest, XGBoost

[CM8]

Kaggle URL: https://www.kaggle.com/c/ece657as21-asg2

Team Name: Group 9

Team Member:

- Ishpinder Kaur B (https://www.kaggle.com/ishpinderkaurb)
- Yuan Sun (https://www.kaggle.com/szsunyuan)