ECE 657A: Data and Knowledge Modelling and Analysis

Assignment 2: Classification using Naive Bayes, Decision tree, Random forest, XGBoost

CM1 (Data Pre-processing and Preparation)

Dataset - DKMA-Covid19-Jan-States

1. Reading the data and observing the feature description

```
#loading the data
df_covid= pd.read_csv("dkmacovid_train.csv")
#describing the data
df_covid.describe()
```

	Day	State ID	Lat	Long_	Active	Incident_Rate	Total_Test_Results	Case_Fatality_Ratio	Testing_Rate	Density Rank 2020 Census	SexRatio
count	1380.000000	1380.000000	1380.000000	1380.000000	1.380000e+03	1380.000000	1.380000e+03	1380.000000	1380.000000	1380.000000	1380.000000
mean	16.500000	25.239130	39.470717	-92.879928	2.610390e+05	7203.192905	5.271097e+06	1.631757	91763.237514	27.173913	97.760870
std	8.658579	14.513405	6.070494	19.632514	4.914059e+05	2305.025102	6.991478e+06	0.656702	40858.185997	15.378197	3.219219
min	2.000000	1.000000	21.094300	-157.498300	9.550000e+02	1232.233261	3.739460e+05	0.439598	30524.071590	1.000000	94.000000
25%	9.000000	12.000000	35.630100	-105.311100	2.731600e+04	6042.134459	1.310515e+06	1.246993	67457.197525	13.000000	95.000000
50%	16.500000	25.500000	39.583950	-88.259400	1.005915e+05	7453.675956	2.919566e+06	1.499993	85438.613770	28.500000	97.000000
75%	24.000000	37.000000	43.326600	-77.209800	2.592418e+05	8621.924085	6.093790e+06	1.817013	104509.453475	41.000000	99.000000
max	31.000000	51.000000	61.370700	-69.381900	3.283336e+06	12811.162350	4.227902e+07	3.928767	235733.711200	52.000000	109.000000

We find the dataset to have sixteen columns (Day, State ID, State, Lat, Long, Active, Incident_Rate, Total_Test_Results, Case_Fatality_Ratio, Testing_Rate, Resident Population 2020 Census, Population Density 2020 Census, Density Rank 2020 Census, SexRatio, Confirmed, Deaths and Recovered).

```
df_covid.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)
```

Our 16 features are found to be a mixture of data types of Boolean, float, integer and object. The three target features, i.e., our labels are of Boolean type which we will convert into a binary format further, for use in our ML models.

2. <u>Checking for Missing and Duplicate values</u>

```
# rows that contain duplicate value
dup_covid = df_covid.duplicated()
print(dup_covid.any())

# Finding missing values in the data sets
print(df_covid.isnull().sum())
```

```
False
Day                                0
State ID                           0
State                              0
Lat                                0
Long_                              0
Active                             0
Incident_Rate                      0
Total_Test_Results                 0
Case_Fatality_Ratio                0
Testing_Rate                       0
Resident Population 2020 Census    0
Population Density 2020 Census     0
Density Rank 2020 Census           0
SexRatio                           0
Confirmed                          0
Deaths                             0
Recovered                          0
```

No duplicate or missing values were found in our dataset

3. Noise reduction

The data was firstly sorted by State IDs and Days in an ascending order for better understanding of the COVID cases for each state for a period of 30 days.
And as mentioned earlier, we encode the labels by converting them into int data type, hence getting values of 1 for TRUE and 0 for FALSE.
We also rounded off the decimal values of the features to their second decimal place for reducing the complexity of our values.
Further noise reduction is done by the Z score normalizing technique, as normalizing helps reduce any unwanted noise while preserving the original data.

```
#sorting rows by State ID and Day
df_covid_sorted = df_covid.sort_values(by=["State ID","Day"])
```

```
#encoding the labels into binary values
df_covid_sorted[['Confirmed', 'Deaths', 'Recovered']] = df_covid_sorted[['Confirmed', 'Deaths', 'Recovered']].astype(int)
```

```
#rounding off the values to 2 decimal places for better understanding of our data
df_covid_sorted = df_covid_sorted.round(2)
```

```
df_covid_sorted.head(5)
```

	Day	State ID	State	Lat	Long_	Active	Incident_Rate	Total_Test_Results	Case_Fatality_Ratio	Testing_Rate	Resident Population 2020 Census	Population Density 2020 Census	Density Rank 2020 Census	SexRatio	Confirmed	Deaths	Recovered
	2	1	Alabama	32.32	-86.9	162449	7535.06	1891468	1.32	38576.31	5,024,279	99.2	29	94	1	0	0
i	3	1	Alabama	32.32	-86.9	164924	7585.56	1900070	1.31	38751.75	5,024,279	99.2	29	94	1	1	0
!	4	1	Alabama	32.32	-86.9	167080	7629.63	1903388	1.30	38819.42	5,024,279	99.2	29	94	1	1	0
B	5	1	Alabama	32.32	-86.9	172570	7741.76	1910881	1.29	38972.24	5,024,279	99.2	29	94	1	1	0
4	6	1	Alabama	32.32	-86.9	167506	7835.40	1921210	1.30	39182.90	5,024,279	99.2	29	94	1	1	1

4. Converting string variables into int and float

```
#converting the string type variables into int and float
df_covid_sorted['Resident Population 2020 Census']=df_covid_sorted['Resident Population 2020 Census'].str.replace(',','')
df_covid_sorted['Resident Population 2020 Census'] = df_covid_sorted['Resident Population 2020 Census'].astype(int)

df_covid_sorted['Population Density 2020 Census']=df_covid_sorted['Population Density 2020 Census'].str.replace(',','')
df_covid_sorted['Population Density 2020 Census'] = df_covid_sorted['Population Density 2020 Census'].astype(float)
```

```
df_covid_sorted.head()
```

	Day	State ID	State	Lat	Long_	Active	Incident_Rate	Total_Test_Results	Case_Fatality_Ratio	Testing_Rate	Resident Population 2020 Census	Population Density 2020 Census	Density Rank 2020 Census	SexRatio	Confirmed	Deaths
0	2	1	Alabama	32.32	-86.9	162449	7535.06	1891468	1.32	38576.31	5024279	99.2	29	94	1	0
46	3	1	Alabama	32.32	-86.9	164924	7585.56	1900070	1.31	38751.75	5024279	99.2	29	94	1	1
92	4	1	Alabama	32.32	-86.9	167080	7629.63	1903388	1.30	38819.42	5024279	99.2	29	94	1	1
138	5	1	Alabama	32.32	-86.9	172570	7741.76	1910881	1.29	38972.24	5024279	99.2	29	94	1	1
184	6	1	Alabama	32.32	-86.9	167506	7835.40	1921210	1.30	39182.90	5024279	99.2	29	94	1	1

This simple conversion is done for the Resident Population and Population density features, to convert them into int and float data type for enabling our ML models to interpret the data.

5. Dealing with outliers and performing normalization

```
#detecting and removing outliers for the columns 5 to 9
for col in df_covid_sorted.columns[5:10]:
  Q1=df_covid_sorted[col].quantile(0.25)
  Q3=df_covid_sorted[col].quantile(0.75)
  IQR=Q3-Q1
  df_final=df_covid_sorted[~((df_covid_sorted[col]<(Q1-1.5*IQR)) | (df_covid_sorted[col]>(Q3+1.5*IQR)))]
  df_final
print(df_final.shape)
```

```
(1252, 17)
```

```
print(df_covid_sorted.shape)
```

```
(1380, 17)
```

Using the IQR method, we detect and remove outliers for 5 features (Active, Incident_Rate, Total_Test_Results, Case_Fatality_Ratio, Testing_Rate). We chose not to remove outliers from the population features as they are a true indicator of the number of people in a state and does not vary for a state as the days vary. These selected 5 features however, vary on a diurnal basis and may consist of certain erroneous values which affect the distribution of our data.

As seen from the code snippet above, we end up dropping 128 rows consisting of outliers.

```
#Z score normalization
df_z = df_final.copy()
columns_to_scale = ['Active', 'Incident_Rate',  'Total_Test_Results', 'Case_Fatality_Ratio',  'Testing_Rate', 'Resident Population 2020 Census','Population Density 2020 Census',
stds = StandardScaler()
df_z[columns_to_scale] = stds.fit_transform(df_z[columns_to_scale])

print(df_z)
print(df_z.shape)
```

```
      Day  State ID    Active  ...  Confirmed  Deaths  Recovered
0       2         1 -0.224294  ...          1       0          0
46      3         1 -0.219430  ...          1       1          0
92      4         1 -0.215193  ...          1       1          0
138     5         1 -0.204403  ...          1       1          0
184     6         1 -0.214356  ...          1       1          1
...   ...       ...       ...  ...        ...     ...        ...
1195   27        51 -0.540736  ...          1       0          1
1241   28        51 -0.541114  ...          1       0          1
1287   29        51 -0.540982  ...          1       0          1
1333   30        51 -0.541340  ...          1       0          1
1379   31        51 -0.540974  ...          1       0          1
```

We perform normalization upon all the features of our dataset apart from our target features and the Day and State ID features.

As mentioned earlier, the Z score normalizing technique helps reduce any unwanted noise while preserving the original data.

It is important in our case to perform normalization as it helps standardize the various features in our dataset, especially the population features which vary immensely for our states. The Active cases feature is another which required to be scaled as it varies significantly state wise and day wise.

Further, as we will be using dimensionality reduction techniques of PCA and LDA, it is important for the data to be normalized.

ECE 657A: Data and Knowledge Modelling and Analysis

Assignment 2: Classification using Naive Bayes, Decision tree, Random forest, XGBoost

CM2 (Representation Learning)

Dataset - DKMA-Covid19-Jan-States

1. We first split the dataset into train and test sets. Df_final is the recent dataframe that we got after pre-processing.

```
# Seperating the dependent and independent variables
X = df_final.drop(['Confirmed', 'Deaths', 'Recovered'],axis=1)
y = df_final.loc[:,['Confirmed', 'Deaths', 'Recovered']]

# Splitting the data into training and testing sets.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=98)
```

2. We now split the y_train desperately for each label and x_train for PCA and LDA separately as follows.

```
# declairing the y_train and y_test variable one for each label.
y_train_confirmed = y_train.loc[:,['Confirmed']]
y_train_deaths = y_train.loc[:,['Deaths']]
y_train_recovered = y_train.loc[:,['Recovered']]
y_test_confirmed = y_test.loc[:,['Confirmed']]
y_test_deaths = y_test.loc[:,['Deaths']]
y_test_recovered = y_test.loc[:,['Recovered']]
```

3. We normalize the x_train and x_test. We are using Z-score Normalization technique here for scaling after splitting the dataset.

```
#normalizing the training data before feeding it into the PCA and LDA algorithms.
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

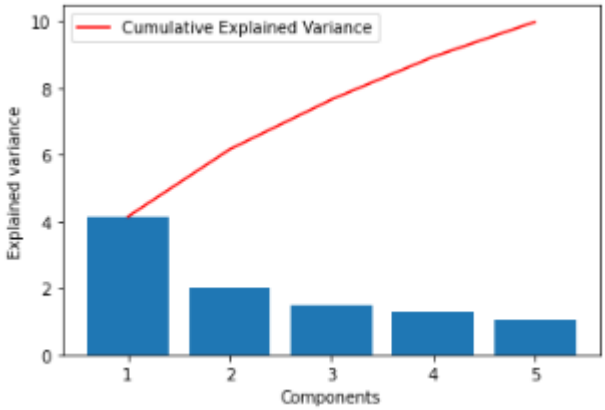
4. Applying the PCA feature reduction technique using sklearn.decomposition. PCA() method. The code snippet is shown below.

```
#applying PCA
pca = PCA(n_components=5)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
```

5. Using scree plot we will now look at the cumulative variance represented by the PCA eigenvectors. Code snippet is show below:

```
#scree plot for pca components. For our case, PCA-5 explains most of the variance than subsequent components
plt.bar(range(1,len(pca.explained_variance_ )+1),pca.explained_variance_ )
plt.ylabel('Explained variance')
plt.xlabel('Components')
plt.plot(range(1,len(pca.explained_variance_ )+1),
         np.cumsum(pca.explained_variance_),
         c='red',
         label="Cumulative Explained Variance")
plt.legend(loc='upper left')
```

Results:



6. Linear discriminant analysis (LDA) is a well-known method for dimensionality reduction. However, the classical Linear Discriminant Analysis (LDA) only works for single-label multi-class classifications and cannot be directly applied to multi-label multi-class classifications. Therefore, LDA method will not provide better better results for one label more than the others.

ECE 657A: Data and Knowledge Modelling and Analysis

Assignment 2: Classification using Naive Bayes, Decision tree, Random forest, XGBoost

CM3 (Decision Trees Classifier)

Dataset - DKMA-Covid19-Jan-States

1. For the decision tree classifier, we make use of our Z-score normalized data, however a non normalized dataset can also be used for decision tree algorithm.

2. We first split our data into train and test set in ratio of 80:20. We use all three of our labels one by one and subsequently create 3 decision trees, one for each label (Confirmed, Deaths, Recovered).

```
x = df_z.drop(['Confirmed','Deaths','Recovered'], axis=1)
y = df_z['Recovered']
#y = df_z['Deaths']
#y = df_z['Confirmed']
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.20, random_state = 98)
```

3. Now we define a function for performing k-fold cross validation on our training data. We also calculate the mean and standard deviation of the accuracy scores for presenting and visualizing the variance of our mean accuracy score.
   For our decision tree classifier we provide a list of input depth parameters (3,5,10) and also run the model once with the "gini impurity" as the splitting rule and once with the "entropy" as the splitting rule

4. Next a function is defined for plotting the mean cross validation accuracy v/s the tree depths we have specified. This is plotted for all three labels of "Recovered", "Deaths" and "Confirmed". The fill_between function is used to give a visual estimate of the variance of the mean accuracy.
   The code snippet for the steps 3 and 4 is given below.

```
# defining a function for performing K-fold cross validation
def running_cross_validation(x, y, tree_depths, cv=5, scoring='accuracy'):
    cv_scores_list = []
    cv_scores_std = []
    cv_scores_mean = []
    accuracy_scores = []
    for depth in tree_depths:
        tree_model = DecisionTreeClassifier(max_depth=depth,criterion="gini")
        #tree_model = DecisionTreeClassifier(max_depth=depth,criterion="entropy")
        cv_scores = cross_val_score(tree_model, x, y, cv=cv, scoring=scoring)
        cv_scores_list.append(cv_scores)
        cv_scores_mean.append(cv_scores.mean())
        cv_scores_std.append(cv_scores.std())
        accuracy_scores.append(tree_model.fit(x, y).score(x, y))
    cv_scores_mean = np.array(cv_scores_mean)
    cv_scores_std = np.array(cv_scores_std)
    accuracy_scores = np.array(accuracy_scores)
    return cv_scores_mean, cv_scores_std, accuracy_scores

# defining a function for plotting cross-validation results
def plot_on_tree_depths(depths, cv_scores_mean, cv_scores_std, accuracy_scores, title):
    fig, ax = plt.subplots(1,1, figsize=(15,5))
    ax.plot(depths, cv_scores_mean, '-o', label='mean cross-validation accuracy', alpha=0.9)
    ax.fill_between(depths, cv_scores_mean-2*cv_scores_std, cv_scores_mean+2*cv_scores_std, alpha=0.2)
    ylim = plt.ylim()
    ax.plot(depths, accuracy_scores, '-*', label='train accuracy', alpha=0.9)
    ax.set_title(title, fontsize=16)
    ax.set_xlabel('Tree depth', fontsize=14)
    ax.set_ylabel('Accuracy', fontsize=14)
    ax.set_ylim(ylim)
    ax.set_xticks(depths)
    ax.legend()

# fitting trees of depth 3,5,10
input_tree_depths = [3,5,10]
final_cv_scores_mean, final_cv_scores_std, final_accuracy_scores = running_cross_validation(x_train, y_train, input_tree_depths)

# plotting mean accuracy vs maximum depth
plot_on_tree_depths(input_tree_depths, final_cv_scores_mean, final_cv_scores_std, final_accuracy_scores,
                    'Accuracy per decision tree depth on training data for "Recovered"')
```

5. Now the best tree depth with the best mean accuracy is obtained using the argmax() function on our list of calculated mean accuracies.

6. The best tree depth and its mean accuracy is printed and further the tree is evaluated on the training and test data, with their respective accuracies also being printed.

   The code snippet for steps 5 and 6 is given below.

```
# calculating the best tree depth with the best mean accuracy
idx_max = sm_cv_scores_mean.argmax()
best_input_tree_depths = input_tree_depths[idx_max]
best_cv_score = final_cv_scores_mean[idx_max]
best_cv_score_std = final_cv_scores_std[idx_max]
print('The depth-{} tree achieves the best mean cross-validation accuracy {} +/- {}% on training dataset'.format(
      best_input_tree_depths, round(best_cv_score*100,5), round(best_cv_score_std*100, 5)))
```

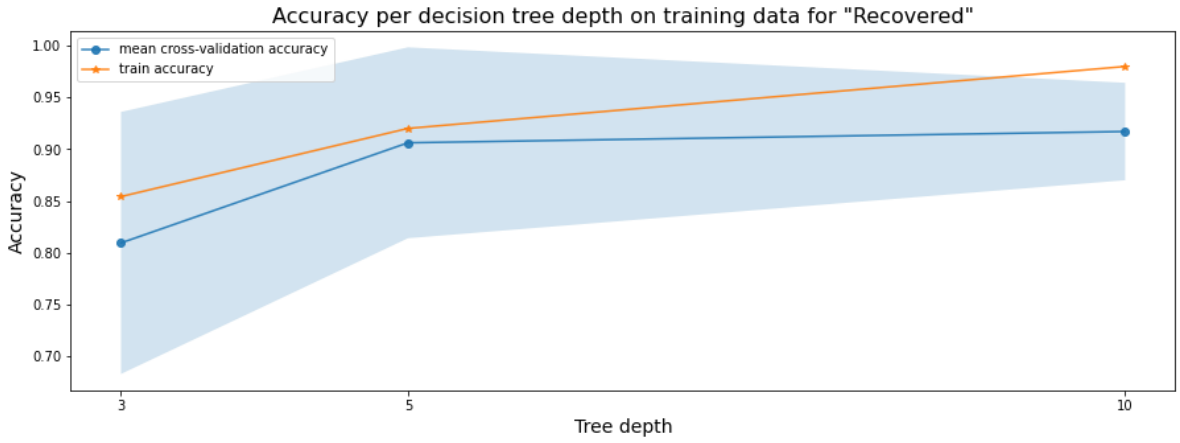
```
The depth-10 tree achieves the best mean cross-validation accuracy 91.70896 +/- 2.35537% on training dataset
```

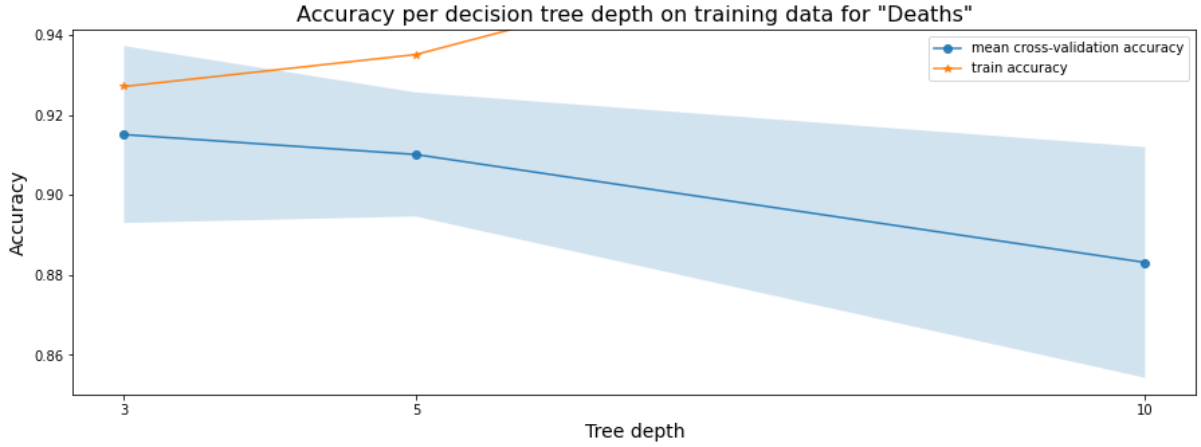
```
# defining a function for training and evaluating a tree
def run_decision_tree(x_train, y_train, x_test, y_test, depth):
    model = DecisionTreeClassifier(max_depth=depth).fit(x_train, y_train)
    accuracy_train = model.score(x_train, y_train)
    accuracy_test = model.score(x_test, y_test)
    print('Single tree depth: ', depth)
    print('Accuracy, Training Set: ', round(accuracy_train*100,5), '%')
    print('Accuracy, Test Set: ', round(accuracy_test*100,5), '%')
    return accuracy_train, accuracy_test


# train and evaluate a 10-depth tree
best_accuracy_train, best_accuracy_test = run_decision_tree(x_train, y_train,
                                                            x_test, y_test,
                                                            best_input_tree_depths)
```

```
Single tree depth:  10
Accuracy, Training Set:  97.8022 %
Accuracy, Test Set:  91.23506 %
```

All the above steps from 1 to 6 have been performed on all three labels and their respective results and graphical plots are shown below.

Graphical plots of mean accuracy v/s max tree depth –





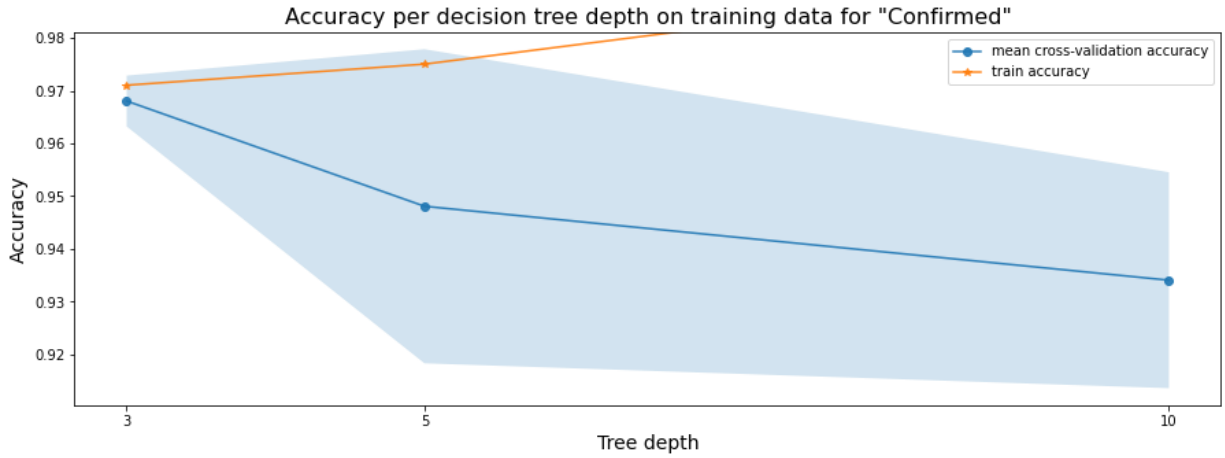


Table consisting of the best tree depths and accuracy scores for the three labels -

	Gini Impurity Splitting				Entropy splitting			
Feature Label	Best Depth	Mean Cross Validation Accuracy	Training accuracy	Test accuracy	Best Depth	Mean Cross Validation Accuracy	Training accuracy	Test accuracy
Recovered	10	91.70896	97.8022	91.23506	10	91.70896	98.002	91.23506
Deaths	3	91.50697	92.70729	93.6255	3	91.30796	92.70729	93.6255
Confirmed	3	96.70398	97.1029	97.60956	3	96.80398	97.1029	97.60956

Analysis-

- We find that the "Recovered" label returns the highest tree depth of 10 as its best depth with a mean validation accuracy of 91%. From its graphical plot we can observe the minimal variance as well at this mean accuracy value for tree depth 10.
  We also get a reasonable overall training accuracy of 97% with a test accuracy of 91%. This suggests that neither is our model for this label overfitting, nor is it underfitting.
  This label being the most balanced of all, seems to provide us with satisfactory results.

- The "Deaths" label is somewhat imbalanced in nature and hence gives us a rather odd value of test accuracy which is higher that the training accuracy.
  From its graphical plot we can see the large variance that exists at its best tree depth value of 3.

- The "Confirmed" label is highly imbalanced in nature and our model thus provides us with quite erroneous results. It gives a max tree depth of 3 with a high mean cross validation accuracy of around 96%.
  The training and test accuracies are almost similar to one another. Further, we can see from its graphical plot that after a tree depth of 3, our mean accuracy has a quite high variance up till the tree depth of 10.

- Another important point to note is regarding the tree splitting rules. We know that gini impurity is much faster as it is less complex, however for our case given a relatively smaller dataset, the speed of computation is not significant enough. Performance wise it can be seen from the table given above, that both the criterions of gini and entropy produce a similar result.

ECE 657A: Data and Knowledge Modelling and Analysis

Assignment 2: Classification using Naive Bayes, Decision tree, Random forest, XGBoost

CM4 (Random Forest)

Dataset - DKMA-Covid19-Jan-States

1. For the random forest classifier, we make use of our Z-score normalized data, however a non normalized dataset can also be used for this algorithm.

2. We first split our data into train and test set in ratio of 80:20. We use all three of our labels one by one and subsequently run the model for each label (Confirmed, Deaths, Recovered).

```
X = df_final.drop(['Confirmed', 'Deaths', 'Recovered'],axis=1)
y = df_final.loc[:,['Confirmed', 'Deaths', 'Recovered']]
print(X.shape)
print(y.shape)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=98)
y_train_confirmed = y_train.loc[:,['Confirmed']]
y_train_deaths = y_train.loc[:,['Deaths']]
y_train_recovered = y_train.loc[:,['Recovered']]
y_test_confirmed = y_test.loc[:,['Confirmed']]
y_test_deaths = y_test.loc[:,['Deaths']]
y_test_recovered = y_test.loc[:,['Recovered']]
```

3. Next we specify the tuned parameters we will be making use of, i.e., the number of trees (estimators) and the tree depths. In CM3 we have made use of cross_val_score and looped the parameters for our decision tree by defining a function. However, we now use the GridSearchCV function for estimating our best hyperparameters. This function helps to loop through our predefined hyperparameters (n_estimators and max_depth) and provides accuracy scores for every combination of hyperparameter. Moreover, having a large number of parameters it is advisable to use GridSearchCV over cross_val_score as the latter requires nested loop to tune multiple hyperparameters.

   The GridSearchCV is implemented for our random forest classifier model, with our k=3 and n_jobs = -1 (this allows use of all processors).

   The code snippet is given below.

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
tuned_param={'max_depth': [3,5,10,None],'n_estimators': [5,10,50,150,200]}
model=GridSearchCV(RandomForestClassifier(n_jobs=-1),param_grid=tuned_param,cv=3,return_train_score=True)
model.fit(x_train, y_train)
```

```
GridSearchCV(cv=3, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=-1,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [3, 5, 10, None],
                         'n_estimators': [5, 10, 50, 150, 200]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

4. Now we print out our best hyper parameters along with the model's best accuracy

```
print("Best hyper paramters:",model.best_params_)
print("Best accuracy value: ",model.best_score_ )
```

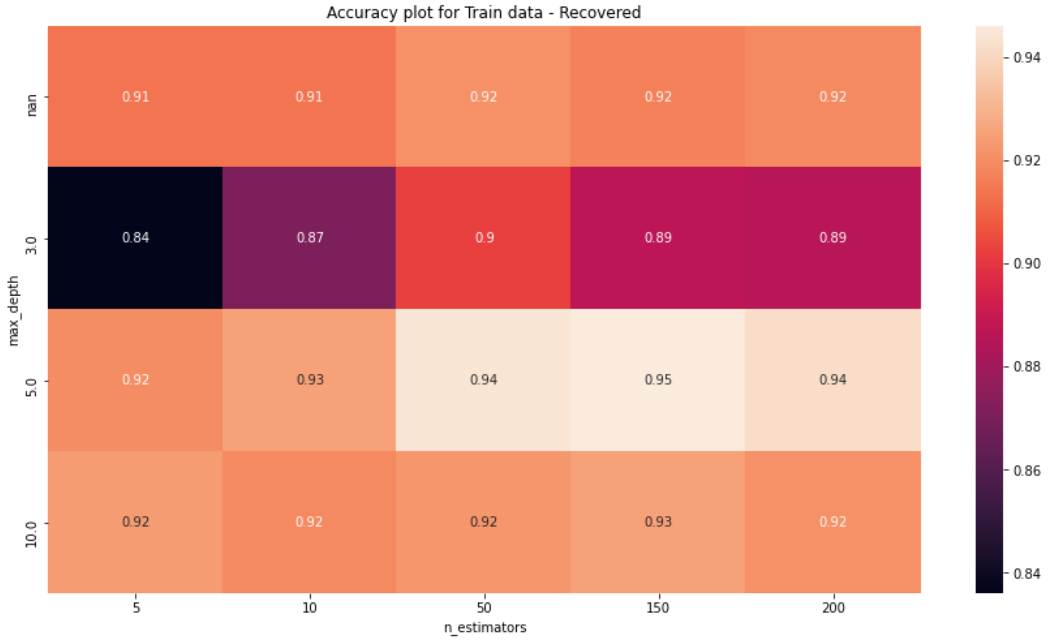
```
Best hyper paramters: {'max_depth': 5, 'n_estimators': 150}
Best accuracy value:  0.9460298621975269
```

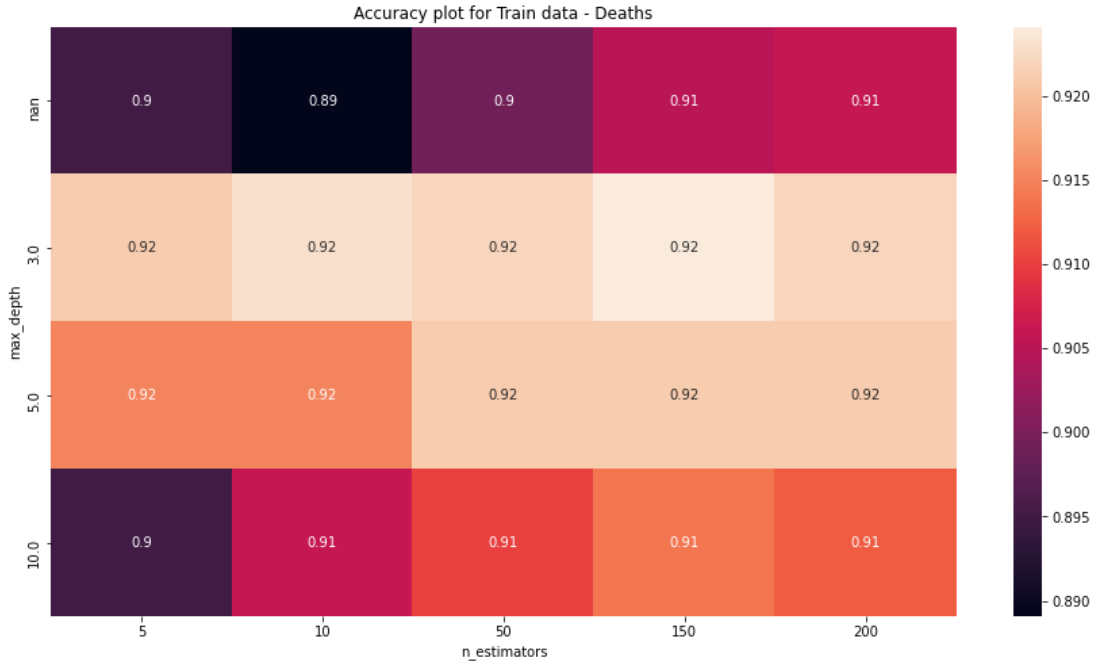
5. We now plot a heat map consisting of all our hyper parameters as the features, with the accuracy score as the values within.

```
plot_df=pd.DataFrame(model.cv_results_['params'])
#Creating a data frame with hyperparameters and accuracy
plot_df["accuracy"]=model.cv_results_['mean_test_score']

#Pivoting the dataframe for plotting heat map
plot_df=plot_df.pivot(index='max_depth',columns='n_estimators',values='accuracy')
#Plotting the graph
plt.figure(figsize=(15,8))
sns.heatmap(data=ac_df,annot=True)
plt.title("Accuracy plot for Train data")
plt.show()
```

All the above steps from 1 to 5 have been performed on all three labels and their respective results and graphical plots are shown below.

<u>Heat maps for the hyperparameters of max depth and n estimators –</u>


Accuracy plot for Train data - Recovered
max_depth
nan
3.0
5.0
10.0
0.91 0.91 0.92 0.92 0.92
0.84 0.87 0.9 0.89 0.89
0.92 0.93 0.94 0.95 0.94
0.92 0.92 0.92 0.93 0.92
5 10 50 150 200
n_estimators
0.94 0.92 0.90 0.88 0.86 0.84


Accuracy plot for Train data - Deaths
max_depth
nan
3.0
5.0
10.0
0.9 0.89 0.9 0.91 0.91
0.92 0.92 0.92 0.92 0.92
0.92 0.92 0.92 0.92 0.92
0.9 0.91 0.91 0.91 0.91
5 10 50 150 200
n_estimators
0.920 0.915 0.910 0.905 0.900 0.895 0.890

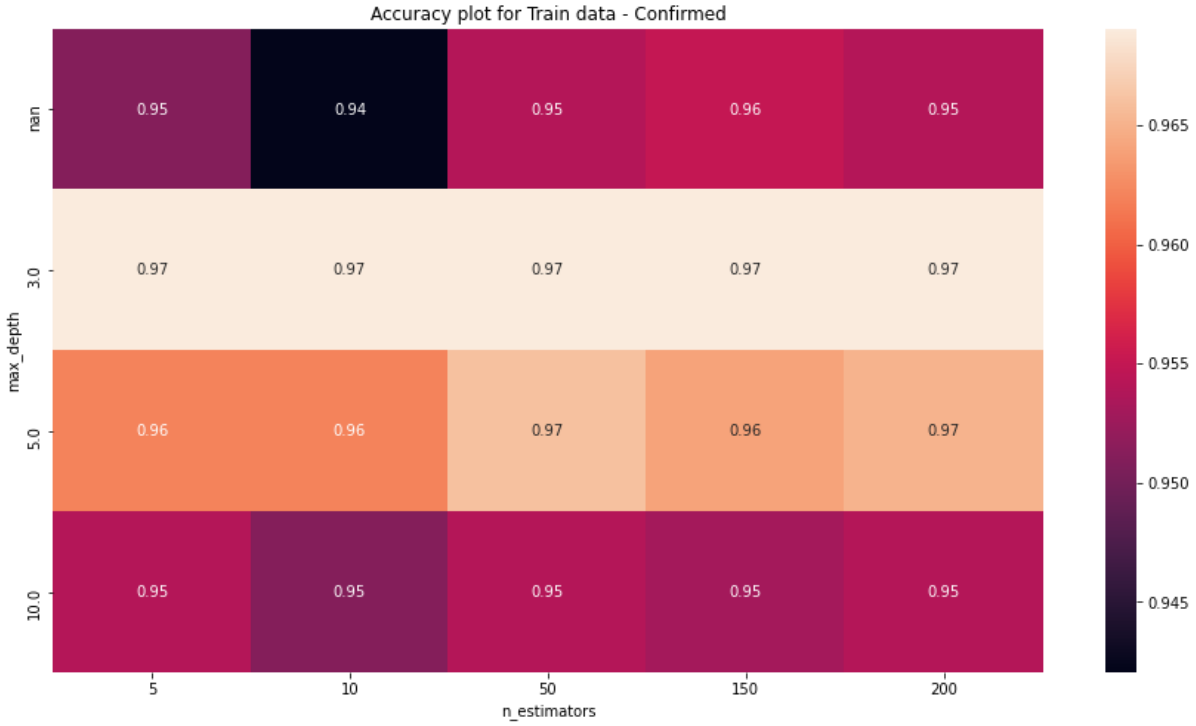

Accuracy plot for Train data - Confirmed
max_depth
nan
3.0
5.0
10.0
0.95 0.94 0.95 0.96 0.95
0.97 0.97 0.97 0.97 0.97
0.96 0.96 0.97 0.96 0.97
0.95 0.95 0.95 0.95 0.95
5 10 50 150 200
n_estimators
0.965 0.960 0.955 0.950 0.945

Table consisting of the best tree depths, best no. of trees and best accuracy scores for the three labels -

Feature Label	Best Tree Depth	Best no. of trees	Best accuracy
Recovered	5	200	94.3
Deaths	3	150	92.4
Confirmed	3	5	96.9

Analysis-

- It can be observed from the obtained results that the "recovered" label utilized the largest number of trees for giving an accuracy of 94%. Being balanced in nature, this label is trained comparatively better than the other labels despite having a lower accuracy than the highly unbalanced label "confirmed".
- The "confirmed" label utilizes a mere 5 trees for obtaining a relatively high accuracy of almost 97%. We know this label is significantly unbalanced and hence can assume that the model is overfitting in this case.
- Even observing the heat maps, we can see how high the average accuracy score is for the various hyperparameter combinations of the "Confirmed" label.
  Whereas, the "Deaths" and "Recovered" labels have much realistic values of accuracy in their heat maps.

Dataset - PCA Feature Set

1. We first split our data into train and test set in ratio of 80:20. We are going to use X_train_pca as independent features and y_train_confirmed, y_train_deaths, y_recovered_deaths as lables seperatelyto train different models.

2. Next, we specify the tuned parameters we will be making use of, i.e., the number of trees (estimators) and the tree depths.

   The GridSearchCV is implemented for our random forest classifier model, with our k=3 and n_jobs = -1 (this allows use of all processors).

   The code snippet is given below.

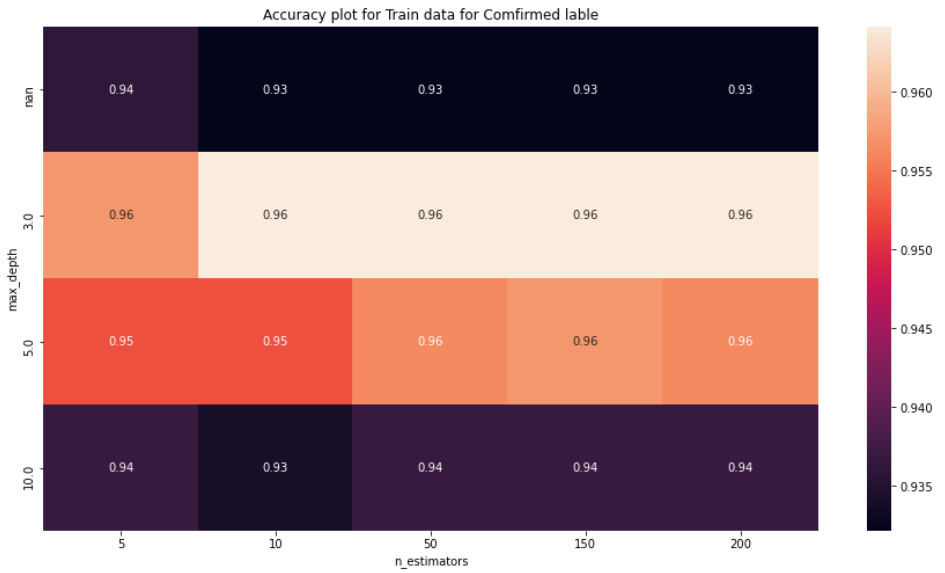
```
#---------Confirmed-----------#
tuned_param={'max_depth': [3,5,10,None],'n_estimators': [5,10,50,150,200]}
model=GridSearchCV(RandomForestClassifier(n_jobs=-1),param_grid=tuned_param,cv=3,return_train_score=True)
model.fit(X_train_pca, y_train_confirmed)

print("Best hyper paramters for Comfirmed lable:",model.best_params_)
print("Best accuracy value for Comfirmed lable: ",model.best_score_ )

ac_df=pd.DataFrame(model.cv_results_['params'])
#Creating a data frame with hyperparameters and accuracy
ac_df["accuracy"]=model.cv_results_['mean_test_score']

#Pivoting the dataframe for plotting heat map
ac_df=ac_df.pivot(index='max_depth',columns='n_estimators',values='accuracy')
#Plotting the graph
plt.figure(figsize=(15,8))
sns.heatmap(data=ac_df,annot=True)
plt.title("Accuracy plot for Train data for Comfirmed lable")
plt.show()
```

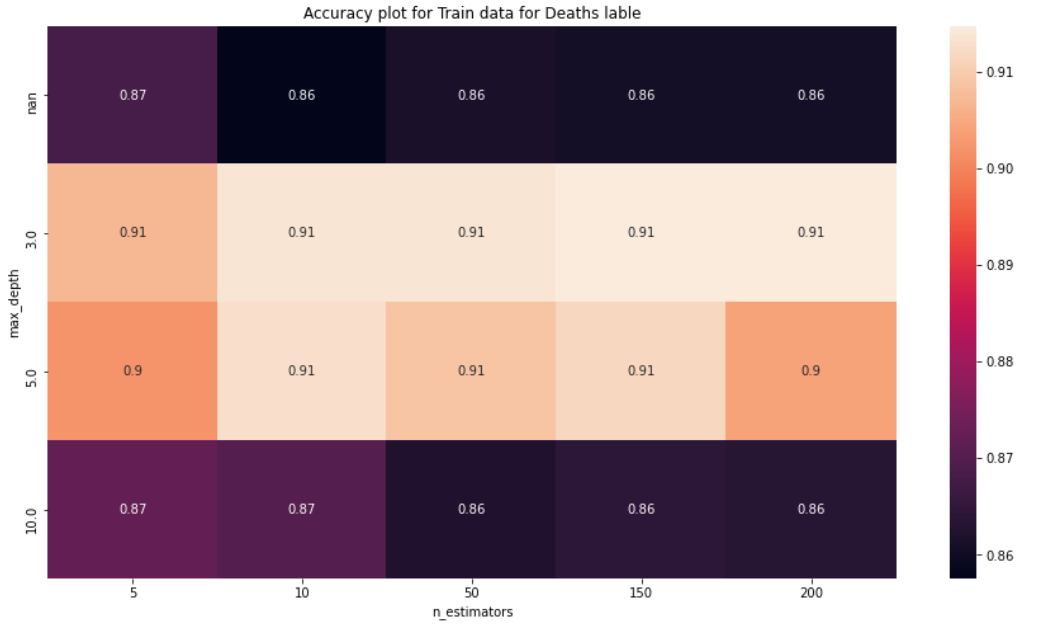
3. Now we print out our best hyper parameters along with the model's best accuracy and the heat map consisting of all our hyper parameters as the features, with the accuracy score as the values within. We used the same code used in the above section so not mentioning the code with this section. Below are the screen shots of the outputs for each lables.

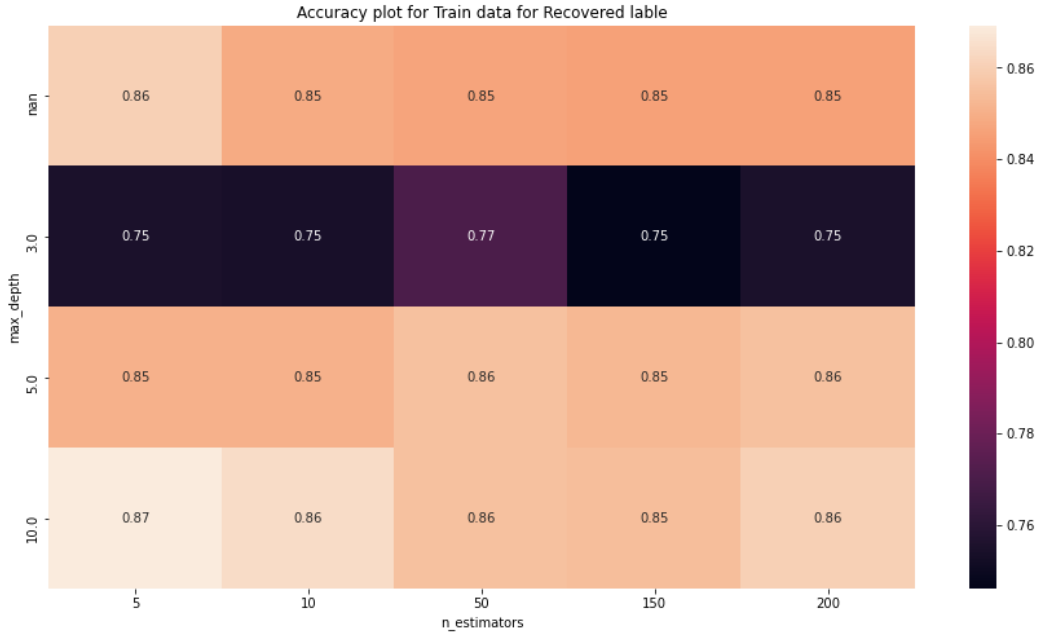
```
Best hyper paramters for Comfirmed lable: {'max_depth': 3, 'n_estimators': 10}
Best accuracy value for Comfirmed lable:  0.9641472868217055
```

```
Best hyper paramters for Deaths lable: {'max_depth': 3, 'n_estimators': 150}
Best accuracy value for Deaths lable:  0.9147286821705426
```

```
Best hyper paramters for Recovered lable: {'max_depth': 10, 'n_estimators': 5}
Best accuracy value for Recovered lable:  0.8691860465116279
```







4. Table consisting of the best tree depths, best no. of trees and best accuracy scores for the three labels -

Feature Label	**Best Tree Depth**	**Best no. of trees**	**Best accuracy**
Confirmed	3	10	96.41
Deaths	3	150	91.47
Recovered	10	5	89.91

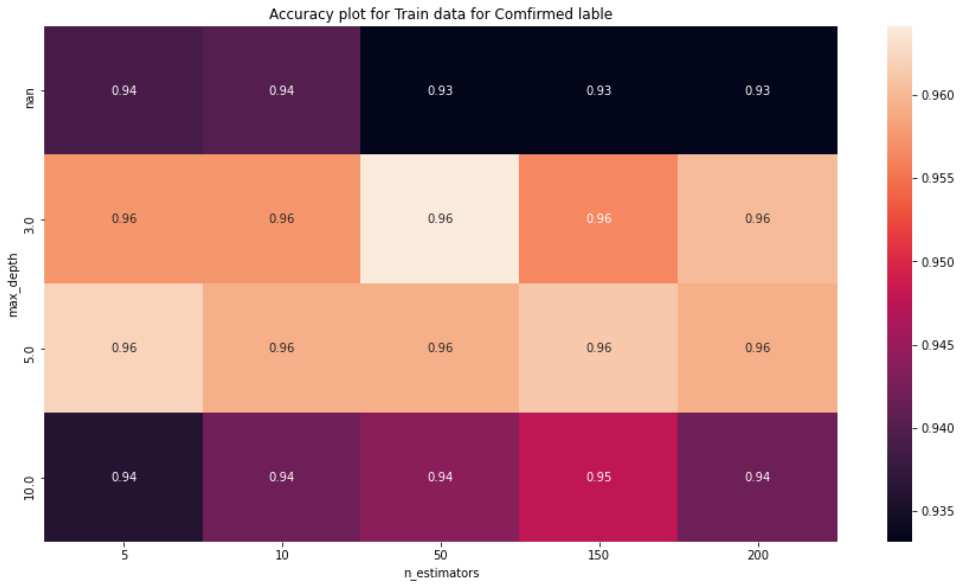
Dataset - LDA Feature Set

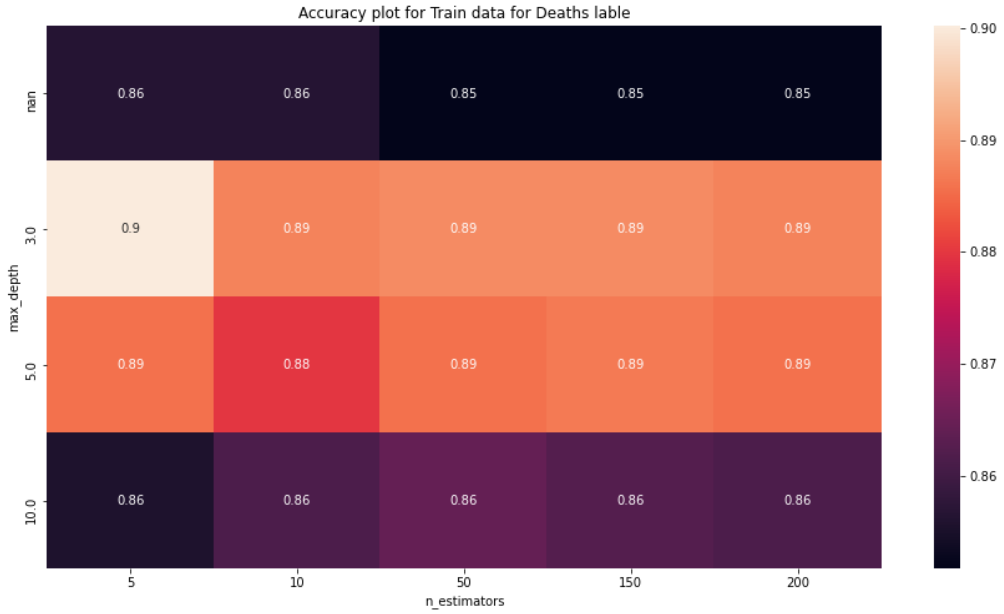
1. We follow the exact same procedure with the LDA dataset as well but using X_train_lda_confirmed variable as the training dataset. Below is the code snippet for the LDA dataset. Rest we use the same variable as used in the above section for y_train.

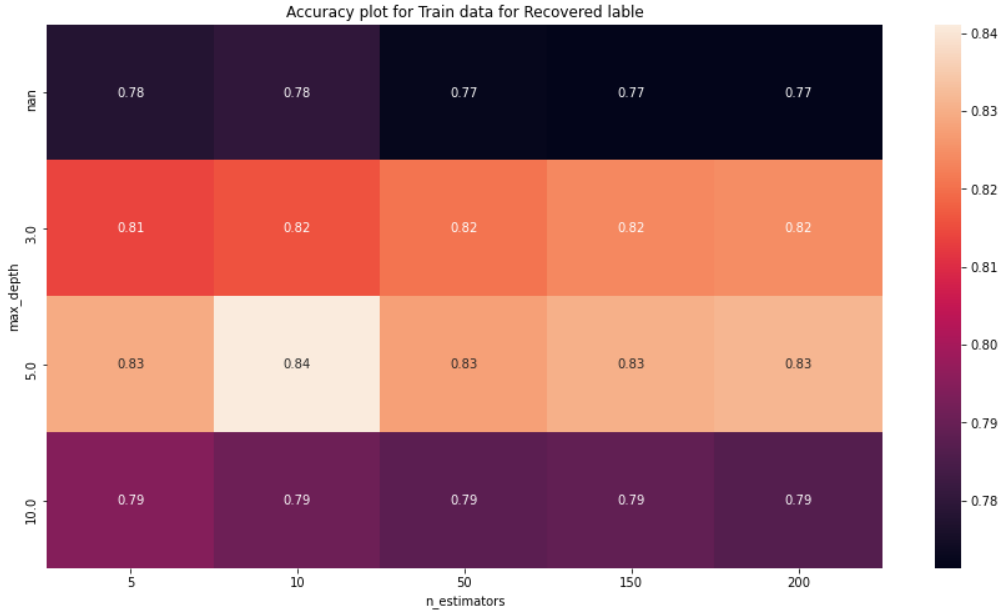
```
#---------Confirmed-----------#
tuned_param={'max_depth': [3,5,10,None],'n_estimators': [5,10,50,150,200]}
model=GridSearchCV(RandomForestClassifier(n_jobs=-1),param_grid=tuned_param,cv=3,return_train_score=True)
model.fit(X_train_lda_confirmed, y_train_confirmed)

print("Best hyper paramters for Comfirmed lable:",model.best_params_)
print("Best accuracy value for Comfirmed lable: ",model.best_score_ )

ac_df=pd.DataFrame(model.cv_results_['params'])
#Creating a data frame with hyperparameters and accuracy
ac_df["accuracy"]=model.cv_results_['mean_test_score']

#Pivoting the dataframe for plotting heat map
ac_df=ac_df.pivot(index='max_depth',columns='n_estimators',values='accuracy')
#Plotting the graph
plt.figure(figsize=(15,8))
sns.heatmap(data=ac_df,annot=True)
plt.title("Accuracy plot for Train data for Comfirmed lable")
plt.show()
```

2. Now we print out our best hyper parameters along with the model's best accuracy and the heat map consisting of all our hyper parameters as the features, with the accuracy score as the values within. We used the same code used in the above section so not mentioning the code with this section. Below are the screen shots of the outputs for each labels.

```
Best hyper paramters for Comfirmed lable: {'max_depth': 3, 'n_estimators': 50}
Best accuracy value for Comfirmed lable:  0.9641472868217055


 Best hyper paramters for Deaths lable: {'max_depth': 3, 'n_estimators': 5}
 Best accuracy value for Deaths lable:  0.9001937984496124


Best hyper paramters for Recovered lable: {'max_depth': 5, 'n_estimators': 10}
Best accuracy value for Recovered lable:  0.8410852713178295
```







3. Table consisting of the best tree depths, best no. of trees and best accuracy scores for the three labels -

Feature Label	**Best Tree Depth**	**Best no. of trees**	**Best accuracy**
Confirmed	3	50	96.41
Deaths	3	5	90.01
Recovered	5	10	84.10

4. Analysis:

- We see the same pattern of accuracy decreasing as we go from Confirmed to Deaths to Recovered labels due to imbalance in data.
- We can also see that accuracy of PCA is not very different than that with original dataset in case of Confirmed and Deaths labels, but it decreases in the case of Recovered label.
- Also, the best value of hyperparameters is changing with different datasets and labels.

ECE 657A: Data and Knowledge Modelling and Analysis

Assignment 2: Classification using Naive Bayes, Decision tree, Random forest, XGBoost

CM5 (Gradient Tree Boosting)

Dataset - DKMA-Covid19-Jan-States

1. For the gradient tree boosting classifier, we make use of our Z-score normalized data, however a non normalized dataset can also be used for this algorithm.

2. We first split our data into train and test set in ratio of 80:20. We use all three of our labels one by one and subsequently run the model for each label (Confirmed, Deaths, Recovered).

```
X = df_final.drop(['Confirmed', 'Deaths', 'Recovered'],axis=1)
y = df_final.loc[:,['Confirmed', 'Deaths', 'Recovered']]
print(X.shape)
print(y.shape)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=98)
y_train_confirmed = y_train.loc[:,['Confirmed']]
y_train_deaths = y_train.loc[:,['Deaths']]
y_train_recovered = y_train.loc[:,['Recovered']]
y_test_confirmed = y_test.loc[:,['Confirmed']]
y_test_deaths = y_test.loc[:,['Deaths']]
y_test_recovered = y_test.loc[:,['Recovered']]
```

3. Now we define a function for performing k-fold cross validation on our training data. We also calculate the mean and standard deviation of the accuracy scores for presenting and visualizing the variance of our mean accuracy score.
   For our gradient tree boosting classifier we provide a list of input estimator parameters (5,10,50,150,200) while keeping the other parameters as default.

4. Next a function is defined for plotting the mean cross validation accuracy v/s the estimators we have specified. This is plotted for all three labels of “Recovered”, “Deaths” and “Confirmed”. The fill_between function is used to give a visual estimate of the variance of the mean accuracy. The code snippet for the steps 3 and 4 is given below.

```
def run_cross_validation(x, y, n_estimators, cv=5, scoring='accuracy'):
    scores_list = []
    scores_std = []
    scores_mean = []
    accuracy_scores = []
    for estimators in n_estimators:
        model = GradientBoostingClassifier(n_estimators = estimators)
        scores = cross_val_score(model, x, y, cv=cv, scoring=scoring)
        scores_list.append(scores)
        scores_mean.append(scores.mean())
        scores_std.append(scores.std())
        accuracy_scores.append(model.fit(x, y).score(x, y))
    scores_mean = np.array(scores_mean)
    scores_std = np.array(scores_std)
    accuracy_scores = np.array(accuracy_scores)
    return scores_mean, scores_std, accuracy_scores

# function for plotting cross-validation results
def plot_graph(estimators, scores_mean, scores_std, accuracy_scores, title):
    fig, ax = plt.subplots(1,1, figsize=(15,5))
    ax.plot(estimators, scores_mean, '-o', label='mean cross-validation accuracy', alpha=0.9)
    ax.fill_between(estimators, scores_mean-2*scores_std, scores_mean+2*scores_std, alpha=0.2)
    ylim = plt.ylim()
    ax.plot(estimators, accuracy_scores, '-*', label='train accuracy', alpha=0.9)
    ax.set_title(title, fontsize=16)
    ax.set_xlabel('No. of Estimators', fontsize=14)
    ax.set_ylabel('Accuracy', fontsize=14)
    ax.set_ylim(ylim)
    ax.set_xticks(estimators)
    ax.legend()
```

5. Now the best number of estimator with the best mean accuracy is obtained using the argmax() function on our list of calculated mean accuracies.

6. The best number of estimator and its mean accuracy is printed.

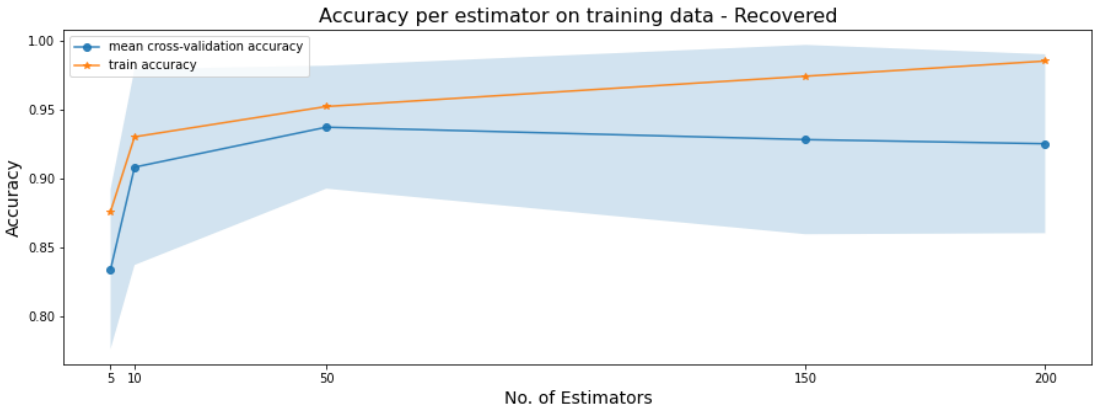
All the above steps from 1 to 6 have been performed on all three labels and their respective results and graphical plots are shown below.

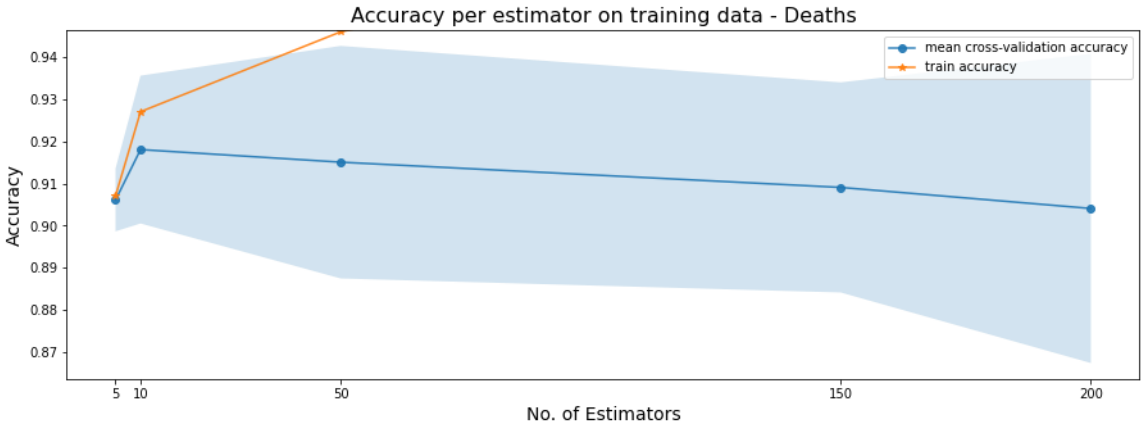
```
#---------Confirmed----------#

# giving input estimators
input_estimators = [5,10,50,150,200]
final_scores_mean, final_scores_std, final_accuracy_scores = run_cross_validation(X_train_lda_confirmed, y_train_confirmed, input

# plotting accuracy
plot_graph(input_estimators, final_scores_mean, final_scores_std, final_accuracy_scores,
                            'Accuracy per estimator on training data - Confirmed')
idx_max = final_scores_mean.argmax()
best_estimator = input_estimators[idx_max]
best_score = final_scores_mean[idx_max]
best_score_std = final_scores_std[idx_max]
print('Total of {} estimators achieve the best mean cross-validation accuracy {} +/- {}% on Confirmed label'.format(
      best_estimator, round(best_score*100,5), round(best_score_std*100, 5)))
```

```
#---------Deaths----------#

# giving input estimators
input_estimators = [5,10,50,150,200]
final_scores_mean, final_scores_std, final_accuracy_scores = run_cross_validation(X_train_lda_deaths, y_train_deaths, input_estir

# plotting accuracy
plot_graph(input_estimators, final_scores_mean, final_scores_std, final_accuracy_scores,
                            'Accuracy per estimator on training data - Deaths')
idx_max = final_scores_mean.argmax()
best_estimator = input_estimators[idx_max]
best_score = final_scores_mean[idx_max]
best_score_std = final_scores_std[idx_max]
print('Total of {} estimators achieve the best mean cross-validation accuracy {} +/- {}% on Deaths label'.format(
      best_estimator, round(best_score*100,5), round(best_score_std*100, 5)))
```

```
#---------Recovered----------#

# giving input estimators
input_estimators = [5,10,50,150,200]
final_scores_mean, final_scores_std, final_accuracy_scores = run_cross_validation(X_train_lda_recovered, y_train_recovered, input

# plotting accuracy
plot_graph(input_estimators, final_scores_mean, final_scores_std, final_accuracy_scores,
                             'Accuracy per estimator on training data - Recovered')
idx_max = final_scores_mean.argmax()
best_estimator = input_estimators[idx_max]
best_score = final_scores_mean[idx_max]
best_score_std = final_scores_std[idx_max]
print('Total of {} estimators achieve the best mean cross-validation accuracy {} +/- {}% on Recovered label'.format(
      best_estimator, round(best_score*100,5), round(best_score_std*100, 5)))
```

Graphical plots of mean accuracy v/s number of estimators–





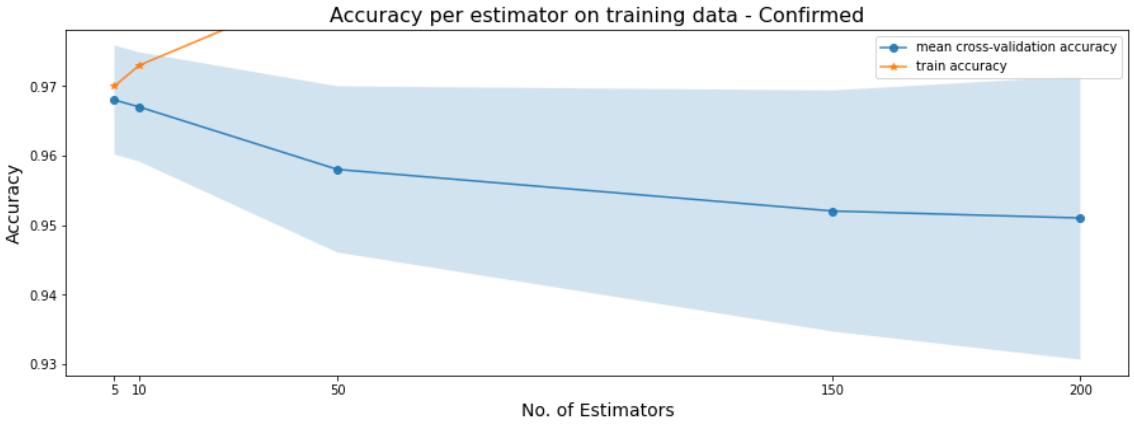


Table consisting of the best estimators and accuracy scores for the three labels –

Feature Labels	**Best no. of estimators**	**Mean Cross Validation Accuracy**
Recovered	50	93.8
Deaths	10	91.8
Confirmed	5	96.8

Analysis –

- As seen from the table and the graph plots above, we find our balanced label of "recovered" to have a reasonably high training accuracy of 93.8% for a total of 50 estimators. Gradient boosting performs better with a higher number of boosting stages, i.e., the n_estimators. The mean cross validation accuracy does not show a great variance either over the 50 estimators point for our "Recovered" label.
- The "Deaths" and "Confirmed" labels being imbalanced in nature present us with odd training and test accuracies. We know that the GradientBoostingClassifier performs much better for a higher number of n_estimators, however for the aforementioned labels we notice that the number of best estimators are found to be quite low, i.e., 5 and 10. The "deaths" label also returns us a higher test accuracy than the training accuracy.

Dataset – PCA and LDA Feature Set

1. We first split our data into train and test set in ratio of 80:20 same as above. We are going to use X_train_pca as independent features and y_train_confirmed, y_train_deaths, y_recovered_deaths as lables seperatelyto train different models.

2. We perform the same procedure with the PCA Features as well as we did with the original dataset.

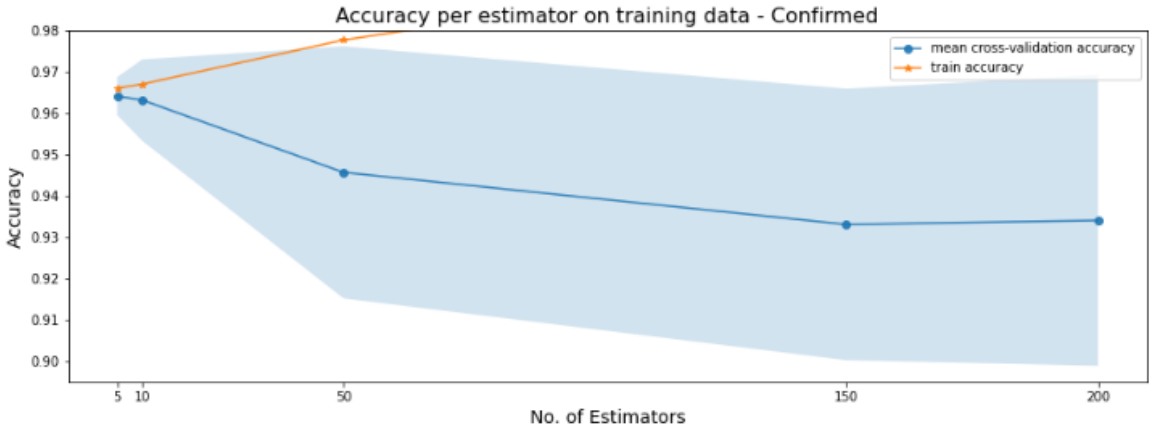
3. The code snippets of the PCA feature dataset is as below

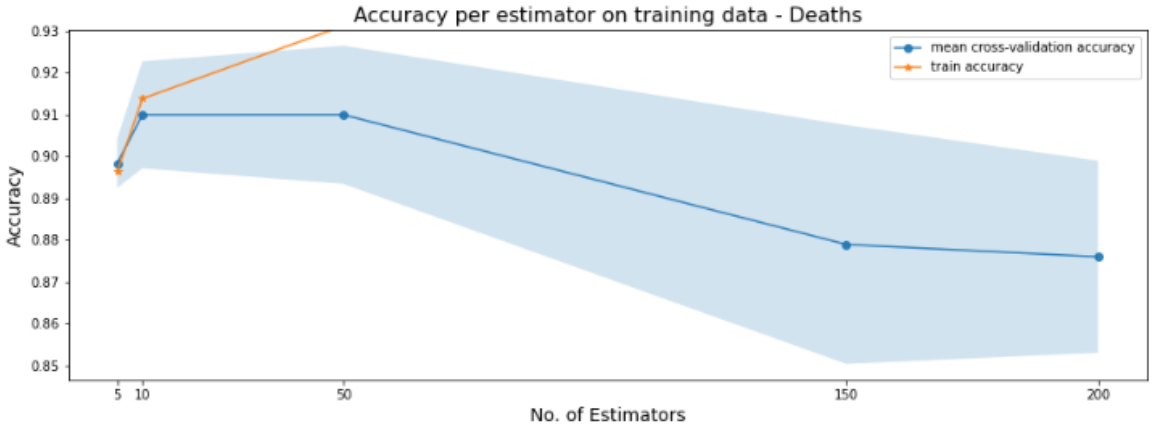
With PCA Feature set.

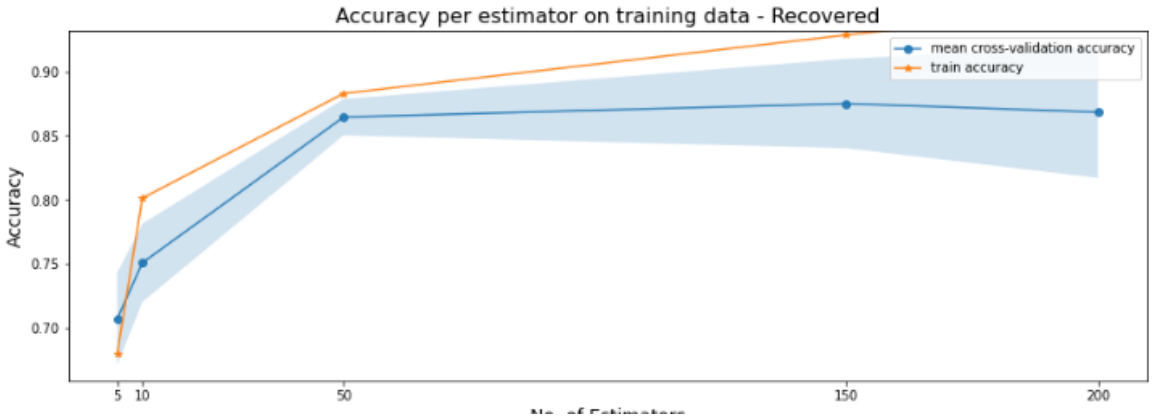
```
#---------Confirmed-----------#

# giving input estimators
input_estimators = [5,10,50,150,200]
final_scores_mean, final_scores_std, final_accuracy_scores = run_cross_validation(X_train_pca, y_train_confirmed, input_estimator

# plotting accuracy
plot_graph(input_estimators, final_scores_mean, final_scores_std, final_accuracy_scores,
                             'Accuracy per estimator on training data - Confirmed')
idx_max = final_scores_mean.argmax()
best_estimator = input_estimators[idx_max]
best_score = final_scores_mean[idx_max]
best_score_std = final_scores_std[idx_max]
print('Total of {} estimators achieve the best mean cross-validation accuracy {} +/- {}% on Confirmed label'.format(
      best_estimator, round(best_score*100,5), round(best_score_std*100, 5)))


#---------Deaths-----------#

# giving input estimators
input_estimators = [5,10,50,150,200]
final_scores_mean, final_scores_std, final_accuracy_scores = run_cross_validation(X_train_pca, y_train_deaths, input_estimators)

# plotting accuracy
plot_graph(input_estimators, final_scores_mean, final_scores_std, final_accuracy_scores,
                             'Accuracy per estimator on training data - Deaths')
idx_max = final_scores_mean.argmax()
best_estimator = input_estimators[idx_max]
best_score = final_scores_mean[idx_max]
best_score_std = final_scores_std[idx_max]
print('Total of {} estimators achieve the best mean cross-validation accuracy {} +/- {}% on Deaths label'.format(
      best_estimator, round(best_score*100,5), round(best_score_std*100, 5)))


#---------Recovered-----------#

# giving input estimators
input_estimators = [5,10,50,150,200]
final_scores_mean, final_scores_std, final_accuracy_scores = run_cross_validation(X_train_pca, y_train_recovered, input_estimator

# plotting accuracy
plot_graph(input_estimators, final_scores_mean, final_scores_std, final_accuracy_scores,
                             'Accuracy per estimator on training data - Recovered')
idx_max = final_scores_mean.argmax()
best_estimator = input_estimators[idx_max]
best_score = final_scores_mean[idx_max]
best_score_std = final_scores_std[idx_max]
print('Total of {} estimators achieve the best mean cross-validation accuracy {} +/- {}% on Recovered label'.format(
      best_estimator, round(best_score*100,5), round(best_score_std*100, 5)))
```

4. The results of the PCA dataset are as below:

```
Total of 5 estimators achieve the best mean cross-validation accuracy 96.41527 +/- 0.22862% on Confirmed label
Total of 50 estimators achieve the best mean cross-validation accuracy 90.9901 +/- 0.82537% on Deaths label
Total of 150 estimators achieve the best mean cross-validation accuracy 87.49683 +/- 1.73693% on Recovered label
```







5. Table consisting of the best estimators and accuracy scores for the three labels –

Feature Labels	**Best no. of estimators**	**Mean Cross Validation Accuracy**
Recovered	150	87.49
Deaths	50	90.99
Confirmed	5	96.41

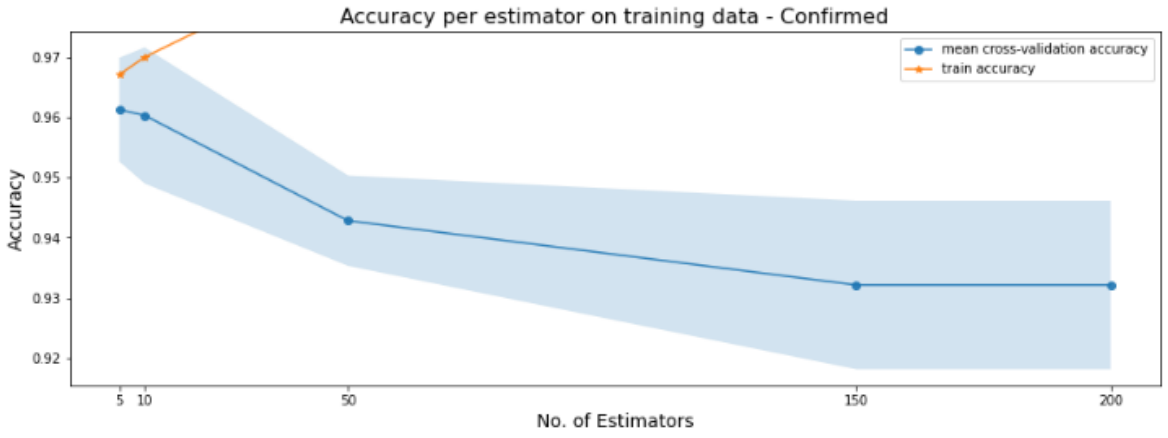
6. The code snippets of the LDA feature dataset is as below

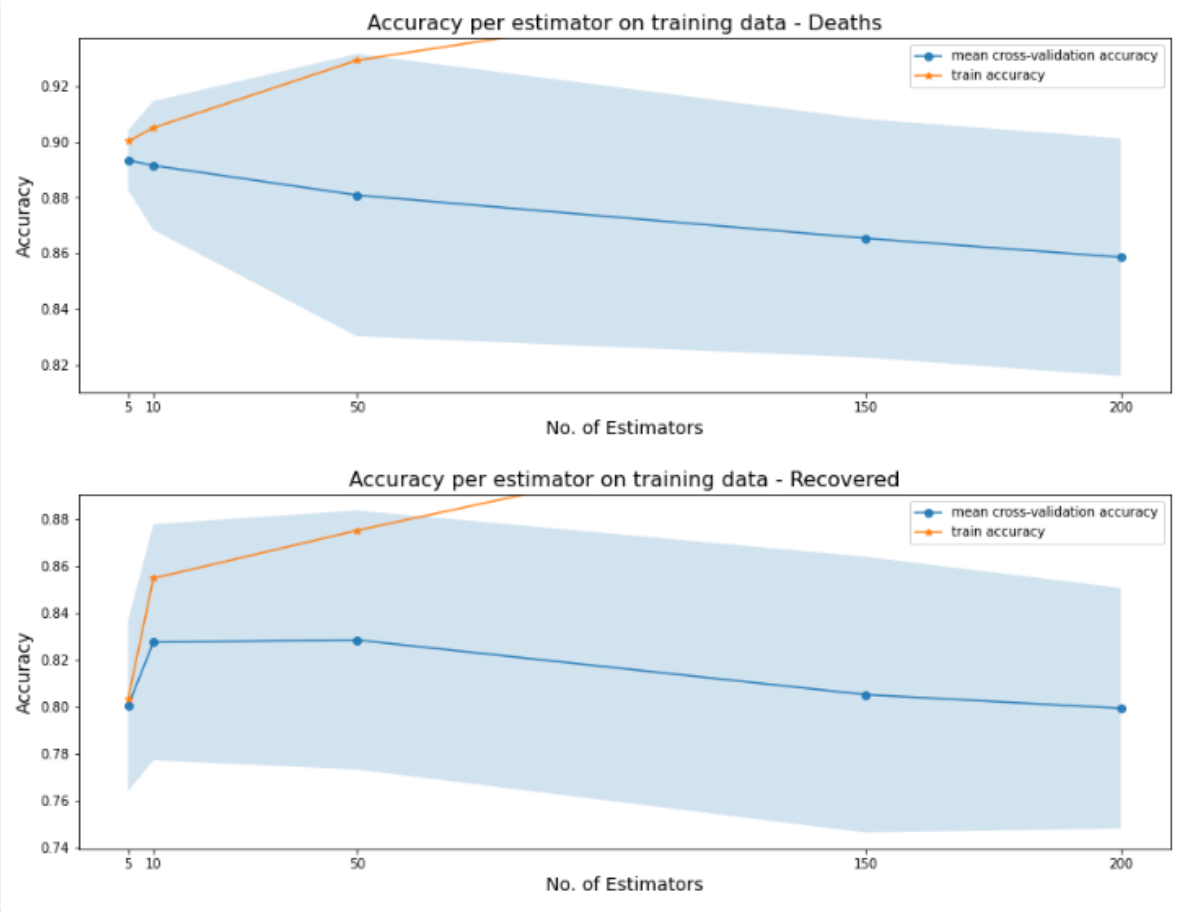
With LDA Feature set.

```
#---------Confirmed----------#

# giving input estimators
input_estimators = [5,10,50,150,200]
final_scores_mean, final_scores_std, final_accuracy_scores = run_cross_validation(X_train_lda_confirmed, y_train_confirmed, input

# plotting accuracy
plot_graph(input_estimators, final_scores_mean, final_scores_std, final_accuracy_scores,
                             'Accuracy per estimator on training data - Confirmed')
idx_max = final_scores_mean.argmax()
best_estimator = input_estimators[idx_max]
best_score = final_scores_mean[idx_max]
best_score_std = final_scores_std[idx_max]
print('Total of {} estimators achieve the best mean cross-validation accuracy {} +/- {}% on Confirmed label'.format(
      best_estimator, round(best_score*100,5), round(best_score_std*100, 5)))


#---------Deaths----------#

# giving input estimators
input_estimators = [5,10,50,150,200]
final_scores_mean, final_scores_std, final_accuracy_scores = run_cross_validation(X_train_lda_deaths, y_train_deaths, input_estir

# plotting accuracy
plot_graph(input_estimators, final_scores_mean, final_scores_std, final_accuracy_scores,
                             'Accuracy per estimator on training data - Deaths')
idx_max = final_scores_mean.argmax()
best_estimator = input_estimators[idx_max]
best_score = final_scores_mean[idx_max]
best_score_std = final_scores_std[idx_max]
print('Total of {} estimators achieve the best mean cross-validation accuracy {} +/- {}% on Deaths label'.format(
      best_estimator, round(best_score*100,5), round(best_score_std*100, 5)))


#---------Recovered----------#

# giving input estimators
input_estimators = [5,10,50,150,200]
final_scores_mean, final_scores_std, final_accuracy_scores = run_cross_validation(X_train_lda_recovered, y_train_recovered, input

# plotting accuracy
plot_graph(input_estimators, final_scores_mean, final_scores_std, final_accuracy_scores,
                             'Accuracy per estimator on training data - Recovered')
idx_max = final_scores_mean.argmax()
best_estimator = input_estimators[idx_max]
best_score = final_scores_mean[idx_max]
best_score_std = final_scores_std[idx_max]
print('Total of {} estimators achieve the best mean cross-validation accuracy {} +/- {}% on Recovered label'.format(
      best_estimator, round(best_score*100,5), round(best_score_std*100, 5)))
```

7. The results of the PCA dataset are as below:

```
Total of 5 estimators achieve the best mean cross-validation accuracy 96.12448 +/- 0.42955% on Confirmed label
Total of 5 estimators achieve the best mean cross-validation accuracy 89.34103 +/- 0.53237% on Deaths label
Total of 50 estimators achieve the best mean cross-validation accuracy 82.84274 +/- 2.75233% on Recovered label
```





8. Table consisting of the best estimators and accuracy scores for the three labels –

Feature Labels	**Best no. of estimators**	**Mean Cross Validation Accuracy**
Recovered	50	82.84
Deaths	5	89.34
Confirmed	5	96.12

9. Analysis:
    a. We find that the recovered label is performing consistently low with all the 3 datasets. It performed well with the original dataset and the lowest with the LDA dataset.
    b. We also find that the Confirmed label is performing best with all the datasets among all the labels.
    c. The Confirmed label always requires the least number of estimators than other labels whereas the Recovered label requires more estimators.

ECE 657A: Data and Knowledge Modelling and Analysis

Assignment 2: Classification using Naive Bayes, Decision tree, Random forest, XGBoost

CM6 (Naive Bayes Classifier)

Dataset - DKMA-Covid19-Jan-States

1. For the decision tree classifier, we make use of our Z-score normalized data, however a non normalized dataset can also be used for decision tree algorithm.

2. We first split our data into train and test set in ratio of 80:20. We use all three of our labels one by one and subsequently create 3 decision trees, one for each label (Confirmed, Deaths, Recovered).

```
x = df_z.drop(['Confirmed','Deaths','Recovered'], axis=1)
y = df_z['Recovered']
#y = df_z['Deaths']
#y = df_z['Confirmed']
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.20, random_state = 98)
```

3. Next we specify the tuned parameters we will be making use of, i.e., the smoothing parameter (var_smoothing = 1e-10, 1e-9, 1e-5, 1e-3, 1e-1).
   We now use the GridSearchCV function for estimating our best hyperparameters.
   The GridSearchCV is implemented for our Naïve Bayes classifier model, with our k=10 and n_jobs = -1 (this allows use of all processors).
   The code snippet is given below.

```
from sklearn.naive_bayes import GaussianNB

#hyperparameters to be tuned.
var_smoothing_list = [1e-10, 1e-9, 1e-5, 1e-3, 1e-1]

accuracy = []

gnb_recovered1 = GaussianNB()

# define grid search
grid = dict(var_smoothing = var_smoothing_list)
#using cv = 10 for 10-fold cross validation as mentioned in the instructions.
grid_search3 = GridSearchCV(estimator = gnb_recovered1, param_grid = grid, n_jobs = -1, cv = 10, scoring = 'accuracy',error_score = 0)
grid_result3 = grid_search3.fit(x_train, y_train)

# summarize results
print("Best: {0} using {1}".format(grid_result3.best_score_, grid_result3.best_params_))
acc = grid_result3.cv_results_['mean_test_score']
params = grid_result3.cv_results_['params']
for mean, param in zip(acc,  params):
    print("{0} with: {1}".format(mean, param))
```

```
Best: 0.7112772277227722 using {'var_smoothing': 0.1}
0.6763564356435643 with: {'var_smoothing': 1e-10}
0.6763564356435643 with: {'var_smoothing': 1e-09}
0.6773465346534653 with: {'var_smoothing': 1e-05}
0.6993168316831684 with: {'var_smoothing': 0.001}
0.7112772277227722 with: {'var_smoothing': 0.1}
```

4. The results obtained from our model are then summarized as shown above, with the best smoothing parameter being printed alongside its accuracy.

All the above steps from 1 to 4 have been performed on all three labels and their respective results are shown below.

Table consisting of the best smoothing factors and best accuracy scores for the three labels -

Feature Label	**Best Smoothing Parameter**	**Accuracy**
Recovered	0.1	71.1
Deaths	0.1	90.7
Confirmed	0.1	96.9

Analysis-

- It can be observed that a smoothing parameter of 0.1 provides the best accuracy for our model for all 3 of our labels.
- However, as opposed to our early models, we find our most balanced label of “Recovered” have the least accuracy amongst all. This is an interesting observation as it suggests our Naïve Bayes model performs much better for our unbalanced label, i.e., the “Confirmed” label with an astonishingly high accuracy of 96.9%, although that high an accuracy could also allude to overfitting.
- More importantly in this observation we find that the highest value of var_smoothing gave us the best accuracies for all our labels.
  This smoothing parameter essentially adds the largest variance amongst all our features to the model’s estimated variance in a certain proportion. Mathematically, this then smoothens our Gaussian curve which then accounts for more samples that are away from the mean of our distribution.
  Hence, we observe a significant improvement in our model’s accuracy.

ECE 657A: Data and Knowledge Modelling and Analysis

Assignment 2: Classification using Naive Bayes, Decision tree, Random forest, XGBoost

CM7 (Interpretability)

Dataset - DKMA-Covid19-Jan-States

The decision tree and naïve bayes models used by us provide significantly different results for our labels of "Recovered", "Deaths" and "Confirmed".

The performance accuracies of both the models is depicted below in Tables 1 and 2.

	Gini Impurity Splitting				Entropy splitting			
Feature Label	Best Depth	Mean Cross Validation Accuracy	Training accuracy	Test accuracy	Best Depth	Mean Cross Validation Accuracy	Training accuracy	Test accuracy
Recovered	10	91.70896	97.8022	91.23506	10	91.70896	98.002	91.23506
Deaths	3	91.50697	92.70729	93.6255	3	91.30796	92.70729	93.6255
Confirmed	3	96.70398	97.1029	97.60956	3	96.80398	97.1029	97.60956

Table 1. Decision Tree Accuracy Scores

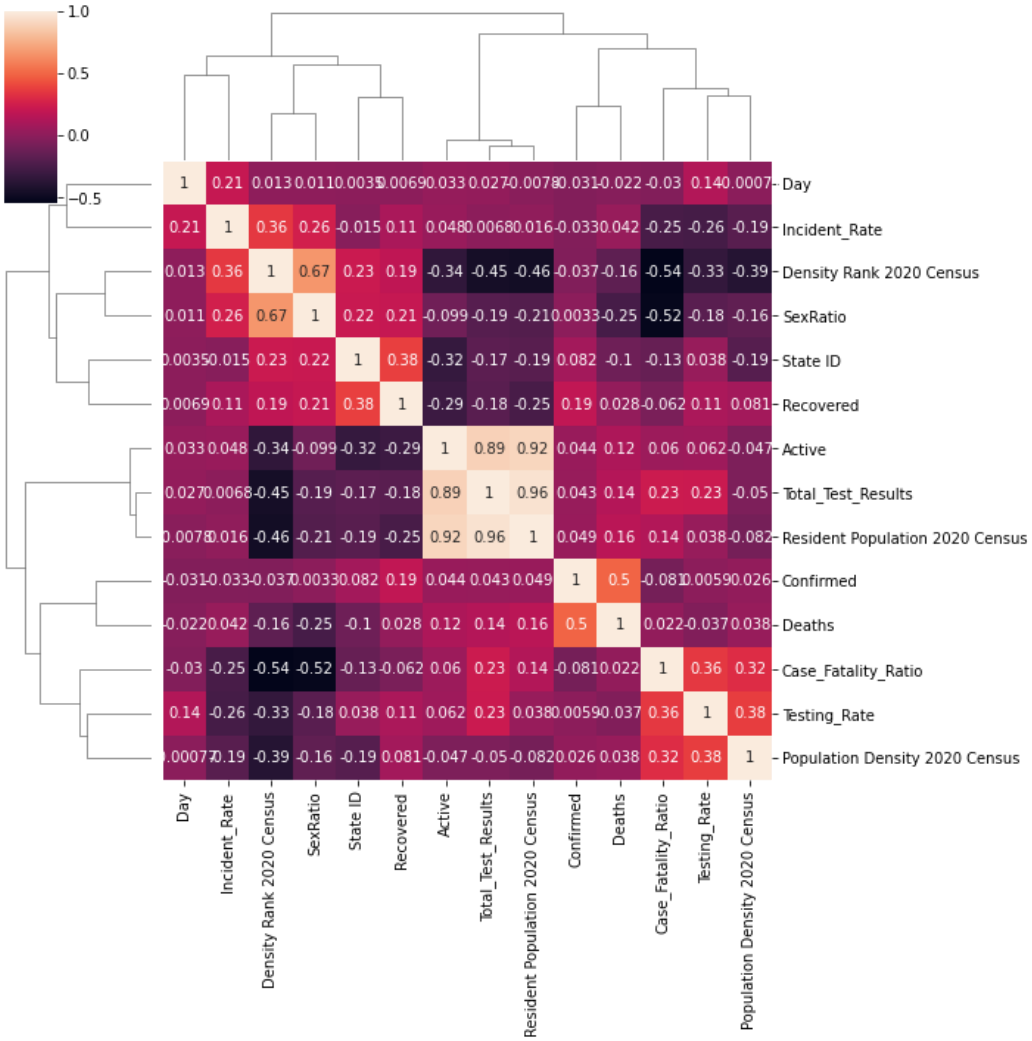
Feature Label	**Best Smoothing Parameter**	**Accuracy**
Recovered	0.1	71.1
Deaths	0.1	90.7
Confirmed	0.1	96.9

Table 2. Naïve Bayes Accuracy Scores

The most striking observation from the above two tables is the considerably poor accuracy of the Naïve Bayes model when compared with that of the Decision Tree for the “Recovered” label.
Both the models however appear to return a high estimate of the accuracy for the “Confirmed” label, which as mentioned earlier is highly imbalanced. This suggests that this label causes both the models to overfit.

The decision tree model is a discriminative machine learning model with a built-in property to estimate suitable features that separate objects representing different classes. Naïve Bayes on the other hand is a generative model and an important feature of it is to assume independence between the training features.
A decision tree model would not depend greatly on the independence of features but the fact that a Naïve Bayes model does, suggests that it’s performance could reduce in the presence of highly correlated features in the training data.

This could verify its poor performance for the “Recovered” label, as being the most balanced label, it could have a high correlation with some of the other features of the data.



Running a simple cluster map for checking the correlation between the features, shows us that the “Recovered” label is significantly correlated with up to 9 features in our data.

Another important point here is to understand the effect the hyperparameters have on our two models.

The splitting rules of Gini impurity or the entropy seem to not effect the performance our decision tree model. This mainly due to the improvement these parameters offer in terms of time not in performance.

The Naïve Bayes model however showed a significant improvement on just the variation of the var_smoothing parameter. As explained earlier, this simply controls the proportion in which variance is added to an estimate variance of our model. However, this smoothing of the gaussian curve tends to immensely improve our model.

The essential understanding from all of the above is that the decision tree model is not as significantly affected by user defined factors as compared to our Naïve Bayes model. Also, the Naïve Bayes model is found to perform worse for features having high correlation values, as was the case for our "Recovered" label feature.

References:

1. Iwendi, C., Bashir, A. K., Peshkar, A., Sujatha, R., Chatterjee, J. M., Pasupuleti, S., Mishra, R., Pillai, S., & Jo, O. (2020). COVID-19 Patient Health Prediction Using Boosted Random Forest Algorithm. Frontiers in public health, 8, 357. https://doi.org/10.3389/fpubh.2020.00357

2. Chaudhary, L., & Singh, B. (2021). Community detection using unsupervised machine learning techniques on COVID-19 dataset. Social network analysis and mining, 11(1), 28. https://doi.org/10.1007/s13278-021-00734-2

3. Maimon, Oded & Rokach, Lior. (2010). Data Mining and Knowledge Discovery Handbook, 2nd ed.

4. Wang H., Ding C., Huang H. (2010) Multi-label Linear Discriminant Analysis. In: Daniilidis K., Maragos P., Paragios N. (eds) Computer Vision – ECCV 2010. ECCV 2010. Lecture Notes in Computer Science, vol 6316. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-15567-3_10

5. https://towardsdatascience.com/comparative-study-on-classic-machine-learning-algorithms-24f9ff6ab222

6. https://towardsdatascience.com/how-to-find-decision-tree-depth-via-cross-validation-2bf143f0f3d6

7. https://towardsdatascience.com/machine-learning-part-18-boosting-algorithms-gradient-boosting-in-python-ef5ae6965be4

8. https://medium.com/@raghavan99o/discriminative-vs-generative-models-c416e53317a7