CM2

February 25, 2021

1 [CM2] Covid dataset (Preprocessing and Algorithms)

1.1 Data Pre-processing

1.1.1 Libraries

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score
from sklearn import tree,preprocessing
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split,KFold,GridSearchCV
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier
from sklearn.preprocessing import StandardScaler
import graphviz

import warnings
warnings.filterwarnings("ignore")
```

1.1.2 Loading dataset

[2]:
```
df_covid = pd.read_csv('covid_train.csv')
df_covid.head()
```

[2]:
```
  Age_Group Client_Gender Case_AcquisitionInfo Reporting_PHU_City  \
0       50s          MALE    NO KNOWN EPI LINK           Oakville
1       20s        FEMALE                   CC             Guelph
2       90s        FEMALE                   OB             Barrie
3       20s        FEMALE  MISSING INFORMATION            Toronto
4       90s        FEMALE                   OB             Ottawa

  Outbreak_Related  Reporting_PHU_Latitude  Reporting_PHU_Longitude  \
0              NaN               43.413997               -79.744796
1              NaN               43.524881               -80.233743
2              Yes               44.410713               -79.686306
3              NaN               43.656591               -79.379358
4              Yes               45.345665               -75.763912
```

```
       Outcome1
0      Resolved
1  Not Resolved
2      Resolved
3      Resolved
4         Fatal
```

1.1.3 Detecting and dropping null values

[3]:
```python
df_covid.isnull().sum()
df_covid = df_covid.dropna(subset=['Age_Group'])
df_covid[['Outbreak_Related']] = df_covid[['Outbreak_Related']].
 ↪fillna(value="No")
df_covid.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14845 entries, 0 to 14850
Data columns (total 8 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Age_Group               14845 non-null  object
 1   Client_Gender           14845 non-null  object
 2   Case_AcquisitionInfo    14845 non-null  object
 3   Reporting_PHU_City      14845 non-null  object
 4   Outbreak_Related        14845 non-null  object
 5   Reporting_PHU_Latitude  14845 non-null  float64
 6   Reporting_PHU_Longitude 14845 non-null  float64
 7   Outcome1                14845 non-null  object
dtypes: float64(2), object(6)
memory usage: 1.0+ MB
```

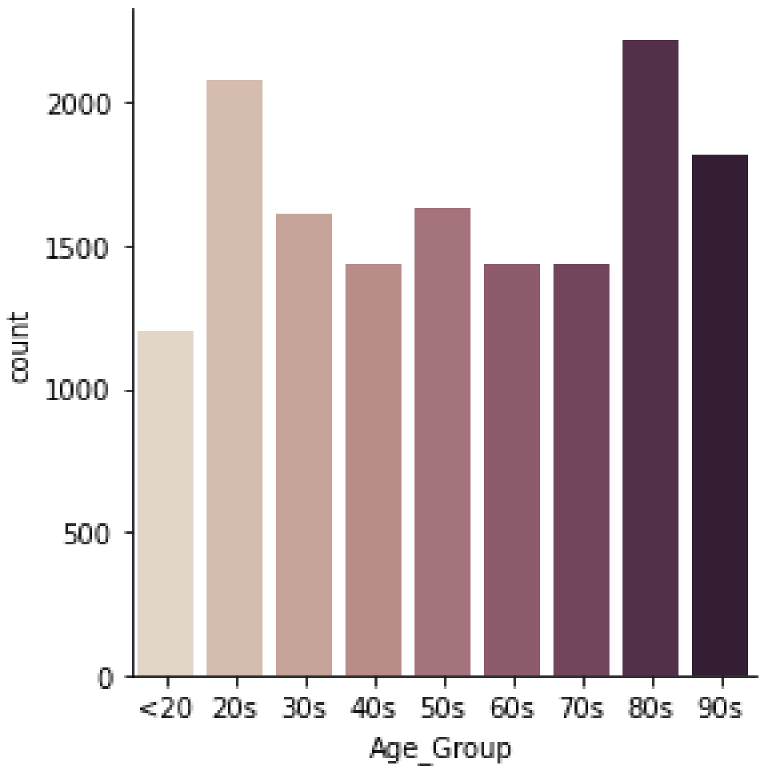
Feature 'age_group' have 6 rows with missing values, we dropped those 6 rows as the data is large and there won't be any data loss due to it. Also, replaced the "NAN" values in feature 'outbreak_related' with "No" as only outbreak related cases are marked "Yes".

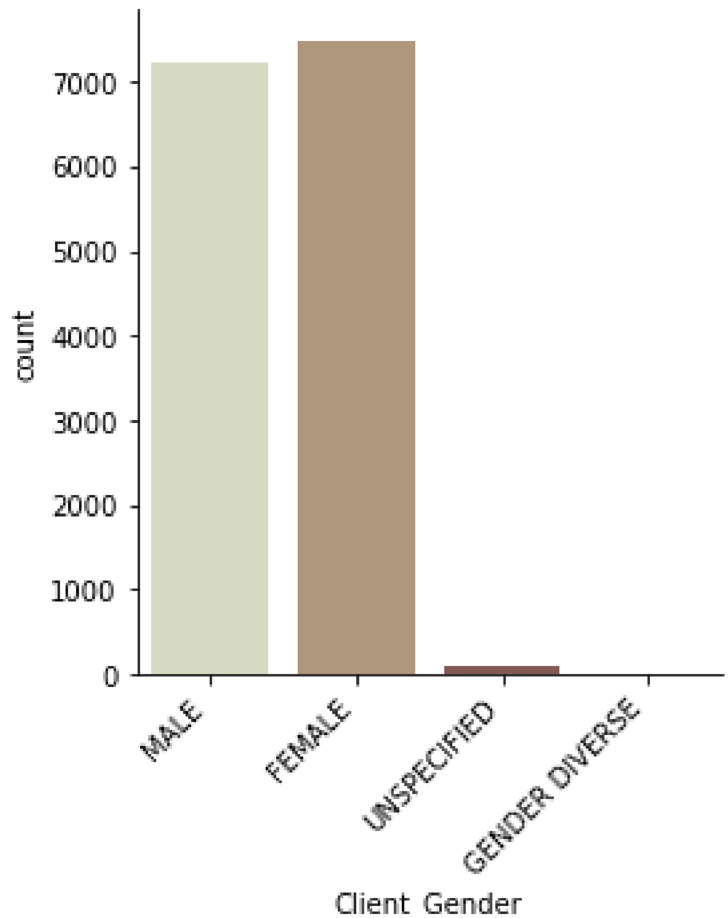
1.1.4 Histograms

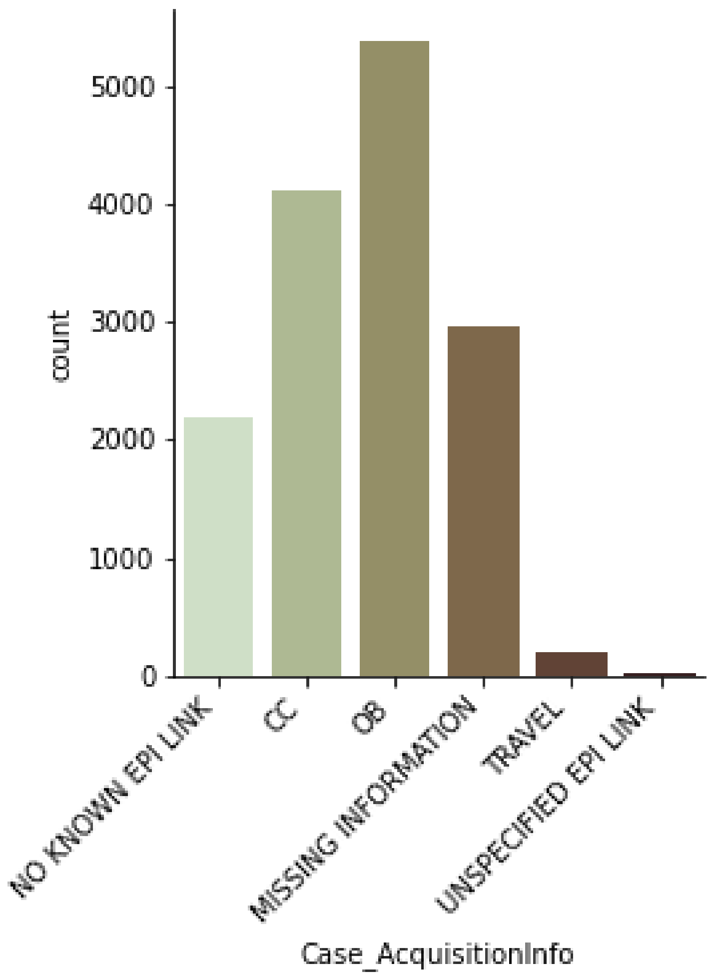
[4]:
```python
sns.catplot(x="Age_Group", kind="count", palette="ch:.25", data=df_covid,␣
 ↪order=['<20','20s','30s','40s','50s','60s','70s','80s','90s'], height=4)

sns.catplot(x="Client_Gender", kind="count", palette="ch:.50", data=df_covid,␣
 ↪height=4)
plt.xticks(
    rotation=45,
    horizontalalignment='right'
)
```

```
sns.catplot(x="Case_AcquisitionInfo", kind="count", palette="ch:.75",␣
 ↪data=df_covid, height=4)
plt.xticks(
    rotation=45,
    horizontalalignment='right'
)
```

[4]:
```
(array([0, 1, 2, 3, 4, 5]),
 [Text(0, 0, 'NO KNOWN EPI LINK'),
  Text(1, 0, 'CC'),
  Text(2, 0, 'OB'),
  Text(3, 0, 'MISSING INFORMATION'),
  Text(4, 0, 'TRAVEL'),
  Text(5, 0, 'UNSPECIFIED EPI LINK')])
```




7000
6000
5000
4000
3000
2000
1000
0
count
MALE
FEMALE
UNSPECIFIED
GENDER DIVERSE
Client_Gender



From the plots,
1. For ‘Age_group’ : We can say that highest number of people are from 80s age group and lowest are from age<20.
2. For ‘Client_Gender’ : We have more females than males and only 2 (which we are not able to see in plot) persons from Gender diversity.
3. For ‘Case_AcquisitionInfo’ : Highest cases can be seen from Outbreak, 2nd highest are from Close contact with COVID positive patient.

Apart from this, we have different city wise cases and longitude/latitude informations of those cities. We also have binary variable ‘Outbreak_Related’, which describes whether a confirmed positive case is linked to an outbreak of COVID-19 in any institutional setting.

1.1.5 One Hot Encoding

[5]:
```
# Chaning datatype of categorical variable from 'object' to 'category'
for col in␣
↪['Client_Gender','Case_AcquisitionInfo','Reporting_PHU_City','Outbreak_Related','Outcome1']
↪
    df_covid[col] = df_covid[col].astype('category')

# One hot encoding
df_covid['Client_Gender'] = df_covid['Client_Gender'].cat.codes
df_covid['Case_AcquisitionInfo'] = df_covid['Case_AcquisitionInfo'].cat.codes
df_covid['Reporting_PHU_City'] = df_covid['Reporting_PHU_City'].cat.codes
df_covid['Outbreak_Related'] = df_covid['Outbreak_Related'].cat.codes

# Dividing dataframe
df_covid_lb = df_covid.copy()
df_covid_num = df_covid.copy()

labelencoder = preprocessing.LabelEncoder()

# Case-1 : Label encoding for age as it is not a categorical variable
df_covid_lb['Age_Group'] = labelencoder.fit_transform(df_covid_lb['Age_Group'])
df_covid_lb['Outcome1'] = labelencoder.fit_transform(df_covid_lb['Outcome1'])

# Case-2 : Changing age to numarical value
df_covid_num['Age_Group'] = df_covid_num['Age_Group'].apply(lambda x: x.
↪strip('s'))
df_covid_num['Age_Group'] = df_covid_num['Age_Group'].replace({"<20": "19"})
df_covid_num['Outcome1'] = labelencoder.fit_transform(df_covid_num['Outcome1'])
```

We have used various data encoding processes for features 'Age_Group' and 'Outcome1' and used that to check the combination of encoding that gives best performance. We created different dataset for encoding process.

Case-1 : Label encoding on age_group feature
Case-2 : Stripping 's' and replacing '<20' with '19' from age_group feature

1.1.6 Seperating X and y

[6]:
```
# Without Standardization
# Case-1 : Label encoding for age as it is not a categorical variable
X1 = df_covid_lb.iloc[:, 0:7]
y1 = df_covid_lb.iloc[:,7]
X_train1, X_test1, y_train1, y_test1 = train_test_split(X1, y1, test_size=0.2,␣
↪random_state=0)

# Case-2 : Changing age to numarical value
X2 = df_covid_num.iloc[:, 0:7]
```

```
y2 = df_covid_num.iloc[:,7]
X_train2, X_test2, y_train2, y_test2 = train_test_split(X2, y2, test_size=0.2,␣
 ↪random_state=0)

# With Standardization
scaler = StandardScaler()

# Case-1 : Label encoding for age as it is not a categorical variable
df_covid_lb[['Reporting_PHU_Latitude','Reporting_PHU_Longitude']] = scaler.
 ↪fit_transform(df_covid_lb[['Reporting_PHU_Latitude','Reporting_PHU_Longitude']])
XX1 = df_covid_lb.iloc[:, 0:7]
yy1 = df_covid_lb.iloc[:,7]
XX_train1, XX_test1, yy_train1, yy_test1 = train_test_split(XX1, yy1,␣
 ↪test_size=0.2, random_state=0)

# Case-2 : Changing age to numarical value
df_covid_num[['Reporting_PHU_Latitude','Reporting_PHU_Longitude']] = scaler.
 ↪fit_transform(df_covid_num[['Reporting_PHU_Latitude','Reporting_PHU_Longitude']])
XX2 = df_covid_num.iloc[:, 0:7]
yy2 = df_covid_num.iloc[:,7]
XX_train2, XX_test2, yy_train2, yy_test2 = train_test_split(XX2, yy2,␣
 ↪test_size=0.2, random_state=0)

max_depth = [3, 5, 10, None]
max_depth1 = list(map(str,max_depth))
```

Splitting the various dataset in train and test model for training the dataset.

1.2 Decision Tree (Without Standardization)

1.2.1 Case-1 : Label encoding for age as it is not a categorical variable

[7]:
```
kf = KFold(random_state=0,n_splits=10)
param_grid = {'max_depth' :[3, 5, 10, None]}

classifier1 = GridSearchCV(DecisionTreeClassifier(random_state=0),␣
 ↪param_grid=param_grid, cv=kf, scoring="accuracy", n_jobs=-1)
classifier1 = classifier1.fit(X_train1, y_train1)

results1 = classifier1.cv_results_
print(results1['mean_test_score']*100)
print(results1['rank_test_score'])
```

```
[63.15261574 65.36736361 65.07287846 64.68519795]
[4 1 2 3]
```

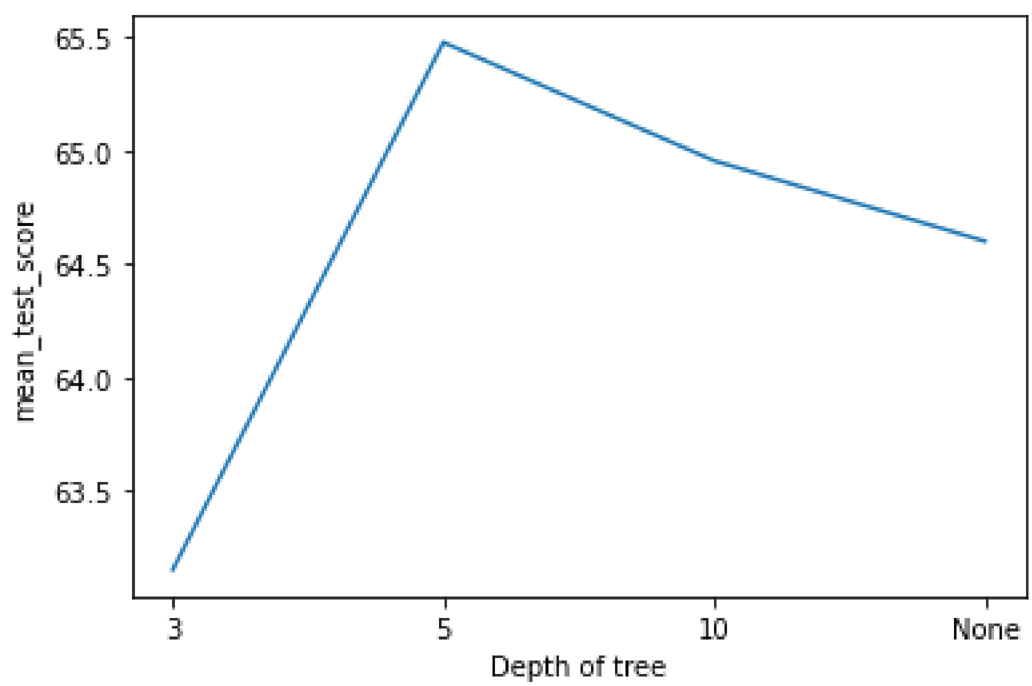
1.2.2 Case-2 : Changing age to numarical value

Applying algorithm on training set

[8]:
```
kf = KFold(random_state=0,n_splits=10)
param_grid = {'max_depth' :[3, 5, 10, None]}

classifier2 = GridSearchCV(DecisionTreeClassifier(random_state=0),␣
↪param_grid=param_grid, cv=kf, scoring="accuracy", n_jobs=-1)
classifier2 = classifier2.fit(X_train2,y_train2)

results2 = classifier2.cv_results_
print(results2['mean_test_score']*100)
print(results2['rank_test_score'])

plt.plot(max_depth1, results2['mean_test_score']*100)
plt.xlabel("Depth of tree")
plt.ylabel("mean_test_score")
```

```
[63.15261574 65.47694014 64.95475678 64.60093777]
[4 1 2 3]
```

[8]:
```
Text(0, 0.5, 'mean_test_score')
```



Applying algorithm on test set

[9]:
```
clf = classifier2.best_estimator_
clf.fit(X_train2, y_train2)
y_pred2 = clf.predict(X_test2)
print(accuracy_score(y_test2, y_pred2)*100)
```

65.67867969013136

1.3 Decision Tree (With Standardization)

1.3.1 Case-1 : Label encoding for age as it is not a categorical variable

[10]:
```
kf = KFold(random_state=0,n_splits=10)
param_grid = {'max_depth' :[3, 5, 10, None]}

classifier1 = GridSearchCV(DecisionTreeClassifier(random_state=0),␣
 ↪param_grid=param_grid, cv=kf, scoring="accuracy", n_jobs=-1)
classifier1 = classifier1.fit(XX_train1, yy_train1)

results1 = classifier1.cv_results_
print(results1['mean_test_score']*100)
print(results1['rank_test_score'])
```

```
[63.15261574 65.36736361 65.07287846 64.68519795]
[4 1 2 3]
```

1.3.2 Case-2 : Changing age to numarical value

Applying algorithm on training set

[11]:
```
kf = KFold(random_state=0,n_splits=10)
param_grid = {'max_depth' :[3, 5, 10, None]}

classifier2 = GridSearchCV(DecisionTreeClassifier(random_state=0),␣
 ↪param_grid=param_grid, cv=kf, scoring="accuracy", n_jobs=-1)
classifier2 = classifier2.fit(XX_train2,yy_train2)

results2 = classifier2.cv_results_
print(results2['mean_test_score']*100)
print(results2['rank_test_score'])
```

```
[63.15261574 65.47694014 64.95475678 64.60093777]
[4 1 2 3]
```

Applying algorithm on test set

[12]:
```
clf = classifier2.best_estimator_
clf.fit(XX_train2, yy_train2)
yy_pred2 = clf.predict(XX_test2)
print(accuracy_score(yy_test2, yy_pred2)*100)
```

65.67867969013136

1.3.3 Observation

The higest accuracy achieved is 65.47 on the training set with 10 fold cross validation. And from the above graphs, it is clear that maximum depth of 5 gives highest accuracy in the cases where feature ‘age_group’ is stripped of ‘s’ and replacing ‘<20’ with ‘19’.

We have used Case-2 for testing accuracy on test set as accuracy increases when the 'age_group' feature is not label encoded. Accuracy of 65.6 is achieved on the testing dataset with the best parameters obtained from GridSearchCV on Case-2.

With or without Standardization, we get same accuracy.

1.3.4 Rules of decision trees

[13]:
```
dot_data = tree.export_graphviz(clf, feature_names= X1.columns,␣
↪class_names=['Fatal','Not resolved','Resolved'], filled=True)
graph = graphviz.Source(dot_data, format="pdf")
graph.view()
```

[13]:
```
'Source.gv.pdf'
```

The first class in the tree is fatal, second is not resolved and third is resolved. First split is done on feature 'Age_Group' with condition less than 65 and gini index 0.667. When the first split condition is true , Resolved and Not Resolved cases are split from Fatal cases. Thus, the false condition,has fatal cases.

For further splits, for fatal cases feature 'Case_AcquisitionInfo' and 'Outbreak_Related' is used, and Resolved/Non-resolved cases are further split on the basis of 'Reporting_PHU_City' and 'Age_Group'.

Overall, we can say that there are higher chances of getting fatal COVID cases, if age is >65.

We have attached 'Source.gv.pdf' (tree) at the end of CM2.

1.4 Random Forest Classifier (Without Standardization)

1.4.1 Case-1 : Label encoding for age as it is not a categorical variable

[14]:
```
kf = KFold(random_state=0,n_splits=10)
param_grid = {'n_estimators': [5, 10, 50, 150, 200], 'max_depth' :[3, 5, 10,␣
↪None]}

rf1 = GridSearchCV(RandomForestClassifier(random_state=0),␣
↪param_grid=param_grid, scoring='accuracy', cv=kf, n_jobs=-1)
rf1 = rf1.fit(X_train1, y_train1)

results1 = rf1.cv_results_
print(results1['mean_test_score']*100)
print(results1['rank_test_score'])
```

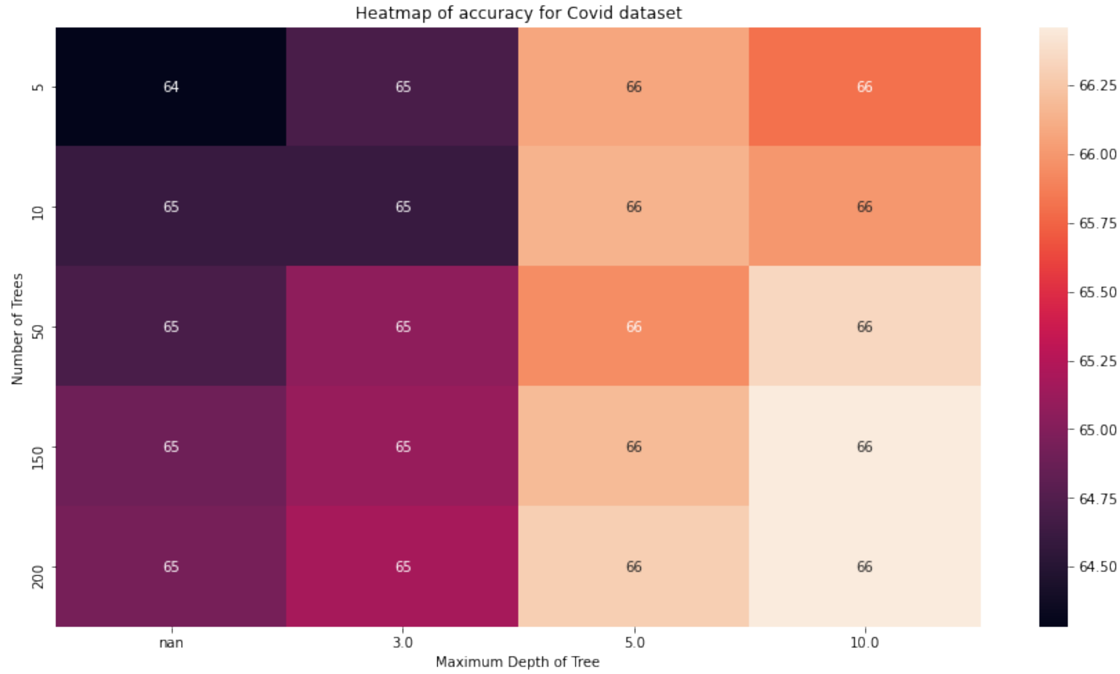
```
[65.46004839 63.54844429 63.19474583 64.02839119 63.97789323 65.5020792
 65.46829571 65.39270123 65.52742392 65.73796091 65.03897441 65.55264098
 65.72947248 65.76319925 65.78003427 64.30635334 64.63469999 64.6768159
 64.77783309 64.7862506 ]
[ 9 19 20 17 18  7  8 10  6  3 11  5  4  2  1 16 15 14 13 12]
```

1.4.2 Case-2 : Changing age to numarical value

Applying algorithm on training set

[15]:
```python
kf = KFold(random_state=0,n_splits=10)
param_grid = {'n_estimators': [5, 10, 50, 150, 200], 'max_depth' :[3, 5, 10,␣
↪None]}

rf2 = GridSearchCV(RandomForestClassifier(random_state=0),␣
↪param_grid=param_grid, scoring='accuracy', cv=kf, n_jobs=-1)
rf2 = rf2.fit(X_train2, y_train2)

results2 = rf2.cv_results_
print(results2['mean_test_score']*100)
print(results2['rank_test_score'])

ac_df = pd.DataFrame(results2['params'])
ac_df["accuracy"] = results2['mean_test_score']*100
ac_df = ac_df.pivot(index='n_estimators',columns='max_depth',values='accuracy')

plt.figure(figsize=(15,8))
sns.heatmap(data=ac_df,annot=True)
plt.title("Heatmap of accuracy for Covid dataset")
plt.xlabel("Maximum Depth of Tree")
plt.ylabel("Number of Trees")
plt.show()
```

```
[64.70195496 64.60923472 65.06395746 65.10615847 65.18194441 66.07476761
 66.14212896 65.94832061 66.18423068 66.29366538 65.81369012 65.99895331
 66.34413498 66.4620581  66.46207228 64.28104408 64.60937655 64.70201169
 64.8956853  64.93781539]
[17 19 13 12 11  7  6  9  5  4 10  8  3  2  1 20 18 16 15 14]
```



Applying algorithm on test set

[16]:
```
clrf = rf2.best_estimator_
clrf.fit(X_train2, y_train2)
y_pred2 = clrf.predict(X_test2)
print(accuracy_score(y_test2, y_pred2)*100)
```

```
67.36274840013473
```

1.5 Random Forest Classifier (With Standardization)

1.5.1 Case-1 : Label encoding for age as it is not a categorical variable

[17]:
```
kf = KFold(random_state=0,n_splits=10)
param_grid = {'n_estimators': [5, 10, 50, 150, 200], 'max_depth' :[3, 5, 10,␣
↪None]}

rf1 = GridSearchCV(RandomForestClassifier(random_state=0),␣
↪param_grid=param_grid, scoring='accuracy', cv=kf, n_jobs=-1)
rf1 = rf1.fit(XX_train1, yy_train1)

results1 = rf1.cv_results_
print(results1['mean_test_score']*100)
print(results1['rank_test_score'])
```

```
[65.46004839 63.54844429 63.19474583 64.02839119 63.97789323 65.5020792
 65.46829571 65.39270123 65.52742392 65.73796091 65.03897441 65.55264098
```

```
 65.72947248 65.76319925 65.78003427 64.30635334 64.63469999 64.6768159
 64.77783309 64.7862506 ]
[ 9 19 20 17 18  7  8 10  6  3 11  5  4  2  1 16 15 14 13 12]
```

1.5.2 Case-2 : Changing age to numarical value

Applying algorithm on training set

[18]:
```
kf = KFold(random_state=0,n_splits=10)
param_grid = {'n_estimators': [5, 10, 50, 150, 200], 'max_depth' :[3, 5, 10,␣
↪None]}

rf2 = GridSearchCV(RandomForestClassifier(random_state=0),␣
↪param_grid=param_grid, scoring='accuracy', cv=kf, n_jobs=-1)
rf2 = rf2.fit(XX_train2, yy_train2)

results2 = rf2.cv_results_
print(results2['mean_test_score']*100)
print(results2['rank_test_score'])

ac_df = pd.DataFrame(results2['params'])
ac_df["accuracy"] = results2['mean_test_score']*100
ac_df = ac_df.pivot(index='n_estimators',columns='max_depth',values='accuracy')
```

```
[64.70195496 64.60923472 65.06395746 65.10615847 65.18194441 66.07476761
 66.14212896 65.94832061 66.18423068 66.29366538 65.81369012 65.99895331
 66.34413498 66.4620581  66.46207228 64.28104408 64.60937655 64.70201169
 64.8956853  64.93781539]
[17 19 13 12 11  7  6  9  5  4 10  8  3  2  1 20 18 16 15 14]
```

Applying algorithm on test set

[19]:
```
clrf = rf2.best_estimator_
clrf.fit(XX_train2, yy_train2)
yy_pred2 = clrf.predict(XX_test2)
print(accuracy_score(yy_test2, yy_pred2)*100)
```

```
67.36274840013473
```

1.5.3 Observation

The highest accuracy achieved by Random Forest classifier on training set 66.46 in Case-2, where feature 'age_group' is not label encoded but 's' is removed from it and replaced '<20' with '19'. We have selected case-2. Accuracy achieved on test set with the best parameters obtained from GridSearchCV is 67.36.

With or withour standardization, we get same accuracy.

1.6 Gradient Tree Boosting (Without Standardization)

1.6.1 Case-1 : Label encoding for age as it is not a categorical variable

[20]:
```
kf = KFold(random_state=0,n_splits=10)
param_grid = {'n_estimators': [5, 10, 50, 150, 200]}

gd1 = GridSearchCV(GradientBoostingClassifier(random_state=0),␣
 ↪param_grid=param_grid, scoring='accuracy', cv=kf, n_jobs=-1)
gd1 = gd1.fit(X_train1, y_train1)

results1 = gd1.cv_results_
print(results1['mean_test_score']*100)
print(results1['rank_test_score'])
```

```
[65.50209339 65.8725914  66.42839516 66.32734251 66.51257024]
[5 4 2 3 1]
```

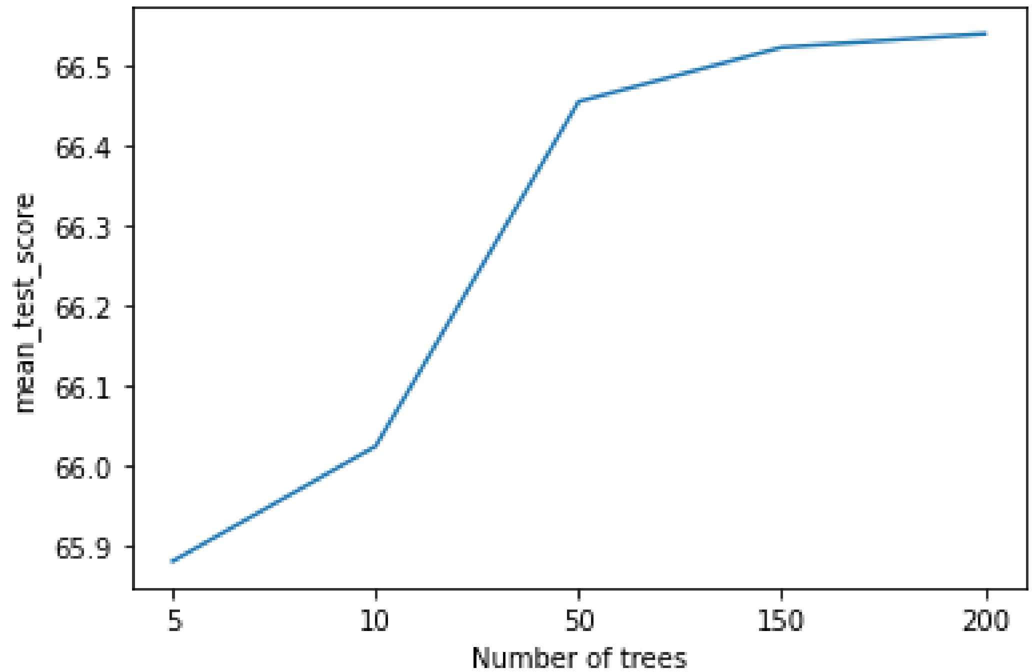
1.6.2 Case-2 : Changing age to numarical value

Applying algorithm on training set

[21]:
```
kf = KFold(random_state=0,n_splits=10)
param_grid = {'n_estimators': [5, 10, 50, 150, 200]}

gd2 = GridSearchCV(GradientBoostingClassifier(random_state=0),␣
 ↪param_grid=param_grid, scoring='accuracy', cv=kf, n_jobs=-1)
gd2 = gd2.fit(X_train2, y_train2)

results2 = gd2.cv_results_
print(results2['mean_test_score']*100)
print(results2['rank_test_score'])

n_estimators = [5, 10, 50, 150, 200]
n_estimators1 = list(map(str,n_estimators))
plt.plot(n_estimators1, results2['mean_test_score']*100)
plt.xlabel("Number of trees")
plt.ylabel("mean_test_score")
```

```
[65.88103728 66.02419874 66.4536335  66.52110121 66.5379575 ]
[5 4 3 2 1]
```

[21]:
```
Text(0, 0.5, 'mean_test_score')
```



Applying algorithm on test set

[22]:
```
clgd = gd2.best_estimator_
clgd.fit(X_train2, y_train2)
y_pred2 = clgd.predict(X_test2)
print(accuracy_score(y_test2, y_pred2)*100)
```

```
67.32906702593466
```

1.7 Gradient Tree Boosting (With Standardization)

1.7.1 Case-1 : Label encoding for age as it is not a categorical variable

[23]:
```
kf = KFold(random_state=0,n_splits=10)
param_grid = {'n_estimators': [5, 10, 50, 150, 200]}

gd1 = GridSearchCV(GradientBoostingClassifier(random_state=0),␣
 ↪param_grid=param_grid, scoring='accuracy', cv=kf, n_jobs=-1)
gd1 = gd1.fit(XX_train1, yy_train1)

results1 = gd1.cv_results_
print(results1['mean_test_score']*100)
print(results1['rank_test_score'])
```

```
[65.50209339 65.8725914  66.42839516 66.32734251 66.51257024]
[5 4 2 3 1]
```

1.7.2 Case-2 : Changing age to numarical value

Applying algorithm on training set

[24]:
```
kf = KFold(random_state=0,n_splits=10)
param_grid = {'n_estimators': [5, 10, 50, 150, 200]}

gd2 = GridSearchCV(GradientBoostingClassifier(random_state=0),␣
 ↪param_grid=param_grid, scoring='accuracy', cv=kf, n_jobs=-1)
gd2 = gd2.fit(XX_train2, yy_train2)

results2 = gd2.cv_results_
print(results2['mean_test_score']*100)
print(results2['rank_test_score'])
```

```
[65.88103728 66.02419874 66.4536335  66.52110121 66.5379575 ]
[5 4 3 2 1]
```

Applying algorithm on test set

[25]:
```
clgd = gd2.best_estimator_
clgd.fit(XX_train2, yy_train2)
yy_pred2 = clgd.predict(XX_test2)
print(accuracy_score(yy_test2, yy_pred2)*100)
```

```
67.32906702593466
```

1.7.3 Observation

The highest accuracy achieved on training set in 66.53 in Case-2. So, we used Case-2 for testing the accuracy on test set. Accuracy on test set from the best parameters achieved from Grid search CV is 67.32.

With or without standardization, we get same accuracy.

1.8 References

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html