```
#%%
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
#%% md
## Loading Covid Dataset, first column is the index (0 based)
#%%
data = pd.read_csv('COVID_Dataset.csv', index_col=0)
#%% md
### Printing summary of dataset
Showing the number of rows, cols, types and overal size of data set in memory
Also showing an overal look of dataset
#%%
data.info()
#%% md
### Input-Output Split
The last column of dataset is the output and the other ones are the inputs
#%%
X = data.iloc[:, 0:-1]
y = data.iloc[:, -1]
#%% md
The output is binary, so we are facing binary classification task
#%%
y.unique()
#%% md
### Normalization
To test the effect of data scaling on classifier quality, we create another dataset based on this one but scale the values of this new set using Standard S
#%% md
#### Standard Scaler: Standardize features by removing the mean and scaling to unit variance.
<img src="z_score.svg">
after applying the above formula, data will have 0 mean and std of 1
#%%
X_scaled = StandardScaler().fit_transform(X)
#%% md
### Train-Test split
We need to split our data into test and train and we have to do it twice, once for normalized data and other one for raw data. To do so we used sklearn tra
random_state is set to get similar results in every experitment during code debugging. (1379 is an arbitrery number)
#%%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=1379)
X_train_s, X_test_s, y_train_s, y_test_s = train_test_split(X_scaled, y, test_size=0.25, random_state=1379)
#%% md
### DeTree
First algorithm is desicion tree, we use this algorithm twice, first without pruning and second with pruning. <br>
pruning is the method of removing unnecceary nodes, which pervent overfitting
#%%
# DeTree withoud pruning
clf_tree = DecisionTreeClassifier()
clf_tree_s = DecisionTreeClassifier()
#%% md
fitting decision tree over trainng data (once for normalized and once for unnormalized)
#%%
clf_tree.fit(X_train, y_train)
clf_tree_s.fit(X_train_s, y_train_s)
#%% md
Get predictions of classifers
#%%
p_train = clf_tree.predict(X_train)
p_train_s = clf_tree_s.predict(X_train_s)

p_test = clf_tree.predict(X_test)
p_test_s = clf_tree_s.predict(X_test_s)
#%% md
Calculating classfication metrics for each case
#%%
print('---- Raw Data ----')
print(f'[Train]:')
print(classification_report(y_train, p_train))
print('[Test]:')
print(classification_report(y_test, p_test))
#%%
print('---- Normalized Data ----')
print(f'[Train]:')
print(classification_report(y_train_s, p_train_s))
print('[Test]:')
print(classification_report(y_test_s, p_test_s))
#%% md
Based on the above results, we can see that normalization didn't have any effect on data which is completely expected because
"The algorithm is based on partitioning the data to make predictions, therefore, it does not require normalization. For example, a decision tree splits a r
Source: https://www.kdnuggets.com/2022/07/random-forest-algorithm-need-normalization.html
#%% md
### DeTree with Pruning
Now let's see the results after pruning trees. <br>
for the pruning pars, we use this algorithm "Minimal Cost-Complexity Pruning", which needs a value alpha.<br>
$R_\alpha(T) = R(T) + \alpha|\widetilde{T}|$<br>
If we set alpha = 0, no pruning will be done. We test pruning with 4 different alphas
#%%
list_alpha = [0.0, 0.00001, 0.0001, 0.001, 0.01]
for alpha in list_alpha:
    clf = DecisionTreeClassifier(ccp_alpha = alpha)
    clf.fit(X_train, y_train)
    p_train = clf.predict(X_train)
```

```
    p_test = clf.predict(X_test)
    print(f'---- Alpha: {alpha} ----')
    print(f'Train: {clf.score(X_train, y_train)}')
    print(f'Test: {clf.score(X_test, y_test)}')
#%% md
So, based on pruning results, we can see that by eliminating some extra nodes, we can increase the test accuracy by sacrificing a portion of trainig accura
but this improvement stops at some point, because we are sacrificing too much nodes, which cause in loosing model complexity which results in both smaller
#%% md
### Ada Boost
The second algorithm is ada boost, ada boost needs a base classifier which in this case we're gonna use the same Decision Tree as above <br>
(Default parameter is Decision Tree)
we test 4 different number of base estimators
#%%
list_n = [50, 100, 500, 1000]
#%%
for n in list_n:
    clf = AdaBoostClassifier(n_estimators = n)
    clf.fit(X_train, y_train)
    p_train = clf.predict(X_train)
    p_test = clf.predict(X_test)
    print(f'---- Number of Estimators: {n} ----')
    print(f'[Train]:')
    print(classification_report(y_train, p_train))
    print('[Test]:')
    print(classification_report(y_test, p_test))

#%% md
Increasing the number of estimators, will slightly increase the metrics. This is important because we want to increase the recall as much as possible, whic
#%% md
The problem is, to get recall to increase by 6% (from 51 to 57) we have to use 950 more estimators which means it's going to incraese the trainig time by a
#%%
```