Decision Tree algorithms on Heart Disease UCI dataset along with Analysis of Pruning

Author: K Krishna Swaroop 181CO125, NITK Surathkal

Dataset

This Heart Disease UCI dataset contains 14 attributes, (13 features and 1 target attribute) they are

- age
- sex
- chest pain type (4 values)
- resting blood pressure
- serum cholestoral in mg/dl
- fasting blood sugar > 120 mg/dl
- resting electrocardiographic results (values 0,1,2)
- maximum heart rate achieved
- exercise induced angina
- oldpeak = ST depression induced by exercise relative to rest
- the slope of the peak exercise ST segment
- number of major vessels (0-3) colored by flourosopy
- thal: 3 = normal; 6 = fixed defect; 7 = reversable defect

The goal is to predict whether or not the patient has heart disease (0- no heart disease, 1- possibility of heart disease)

Dataset Exploration

1) Import Libraries

```
import numpy as np
import pandas as pd
import os
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn import tree
from sklearn.metrics import accuracy_score,confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

2) Import Data

```
data_path = '/content/drive/MyDrive/Colab Notebooks/ML-Lab/Decision-Trees/heart.csv
df = pd.read_csv(data_path)
df.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	c
0	63	1	3	145	233	1	0	150	0	2.3	0	
1	37	1	2	130	250	0	1	187	0	3.5	0	
2	41	0	1	130	204	0	0	172	0	1.4	2	
3	56	1	1	120	236	0	1	178	0	0.8	2	
4	57	0	0	120	354	0	1	163	1	0.6	2	

```
X = df.drop(columns = ['target'])
y = df['target']
tmp = df.columns.tolist()
feature_names = tmp[:-1]
class_names = ['Not heart disease','heart disease']
print(feature_names)
print(class_names)
```

```
['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang'
['Not heart disease', 'heart disease']
```

3) Split data

```
x_train,x_test,y_train,y_test = train_test_split(X,y,stratify=y)
print("Training set shape: ", x_train.shape)
print("Testing set shape: ", x_test.shape)
```

```
Training set shape:  (227, 13)
Testing set shape:  (76, 13)
```

Now, we will fit the three decision tree algorithms, without pruning and compare results with pruning

Gini

1) Install required libraries

```
!pip install graphviz
import graphviz
```

```
Requirement already satisfied: graphviz in /usr/local/lib/python3.6/dist-pack
```

2) Fit the model

```
clf = tree.DecisionTreeClassifier(criterion='gini', random_state = 10)
clf = clf.fit(x_train, y_train)
```

```
dot_data = tree.export_graphviz(clf, out_file=None,
                      feature_names=feature_names,
                      class_names=class_names,
                      filled=True, rounded=True,
                      special_characters=True)
graph = graphviz.Source(dot_data)
graph
graph.render("CART_without_pruning")
```

```
'CART_without_pruning.pdf'
```

3) Accuracy

```
y_train_pred = clf.predict(x_train)
y_test_pred = clf.predict(x_test)
print(f'Train score {accuracy_score(y_train_pred,y_train)}')
print(f'Test score {accuracy_score(y_test_pred,y_test)}')
```

```
Train score 1.0
Test score 0.8026315789473685
```

We have achieved a score of **77.6%** on the testing dataset without pruning

C4.5

1) Install required libraries

```
!pip install chefboost
```

```
Requirement already satisfied: chefboost in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: pandas>=0.22.0 in /usr/local/lib/python3.6/dis
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.6/dist
```

```
Requirement already satisfied: tqdm>=4.30.0 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/pytho
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/dist-pack
```

2) Fit model

```
from chefboost import Chefboost as chef
config = {'algorithm': 'C4.5'}
```

```
df['Decision'] = pd.Series(df.target)
df['Decision'] = df['Decision'].replace({0:"Does not have heart disease" , 1: "Has
print(df['Decision'].dtypes)
train=df.sample(frac=0.75,random_state=10) #random state is a seed value
test=df.drop(train.index)
```

```
object
```

```
model = chef.fit(train, config ,test)
```

```
C4.5  tree is going to be built...
-------------------------
finished in  0.40393543243408203  seconds
-------------------------
Evaluate  train set
-------------------------
Accuracy:  100.0 % on  227  instances
Labels:  ['Does not have heart disease' 'Has Heart Disease']
Confusion matrix:  [[108, 0], [0, 119]]
Precision:  100.0 %, Recall:  100.0 %, F1:  100.0 %
-------------------------
Evaluate  validation set
-------------------------
Accuracy:  100.0 % on  76  instances
Labels:  ['Has Heart Disease' 'Does not have heart disease']
Confusion matrix:  [[46, 0], [0, 30]]
Precision:  100.0 %, Recall:  100.0 %, F1:  100.0 %
```

3) Analysis

C4.5 gives 100% accuracy on the test dataset without pruning

ID3

1) Fit the model

```
id3 = tree.DecisionTreeClassifier(criterion='entropy', random_state=10)
id3 = id3.fit(x_train, y_train)
```

```
dot_data = tree.export_graphviz(id3, out_file=None,
                      feature_names=feature_names,
                      class_names=class_names,
                      filled=True, rounded=True,
                      special_characters=True)
graph = graphviz.Source(dot_data)
graph
graph.render("ID3_without_pruning")
```

```
'ID3_without_pruning.pdf'
```

```
y_train_pred = id3.predict(x_train)
y_test_pred = id3.predict(x_test)
print(f'Train score {accuracy_score(y_train_pred,y_train)}')
print(f'Test score {accuracy_score(y_test_pred,y_test)}')
```

```
Train score 1.0
Test score 0.8552631578947368
```

We have achieved a score of **80.2%** on the testing dataset without pruning

Pruning Analysis

Pruning the tree is nothing but stoping the growth of decision tree on an early stage. For that we can limit the growth of trees by setting constrains. We can limit parameters like max_depth , min_samples etc.

An effective way to do is that we can grid search those parameters and choose the optimum values that gives better performace on test data.

As of now we will control these parameters

- max_depth: maximum depth of decision tree
- min_sample_split: The minimum number of samples required to split an internal node:
- min_samples_leaf: The minimum number of samples required to be at a leaf node.

1) Pruning Gini

1) Parameters to prune

```
params = {'max_depth': [2,4,6,8,10,12],
```

```
          'min_samples_split': [2,3,4],
          'min_samples_leaf': [1,2]}

clf = tree.DecisionTreeClassifier(criterion = "gini", random_state=10)
gcv = GridSearchCV(estimator=clf,param_grid=params)
gcv.fit(x_train,y_train)
```

```
GridSearchCV(cv=None, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=Non
                                              criterion='gini', max_depth=Non
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=10,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [2, 4, 6, 8, 10, 12],
                         'min_samples_leaf': [1, 2],
                         'min_samples_split': [2, 3, 4]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

2) Fitting the model

```
model = gcv.best_estimator_
model.fit(x_train,y_train)
y_train_pred = model.predict(x_train)
y_test_pred = model.predict(x_test)
```

3) Accuracy

```
print(f'Train score {accuracy_score(y_train_pred,y_train)}')
print(f'Test score {accuracy_score(y_test_pred,y_test)}')
```

```
Train score 0.775330396475771
Test score 0.75
```

We get an **81.5%** accuracy on CART with pruning!

4) Visualisation

```
dot_data = tree.export_graphviz(model, out_file=None,
                      feature_names=feature_names,
```

```
                        class_names=class_names,
                        filled=True, rounded=True,
                        special_characters=True)
graph = graphviz.Source(dot_data)
graph
graph.render("CART_With_pruning")
```

```
'CART_With_pruning.pdf'
```

2) Pruning ID3

1) Parameters to Prune

```
params = {'max_depth': [2,4,6,8,10,12],
         'min_samples_split': [2,3,4],
         'min_samples_leaf': [1,2]}

clf = tree.DecisionTreeClassifier(criterion = "entropy", random_state=10)
gcv = GridSearchCV(estimator=clf,param_grid=params)
gcv.fit(x_train,y_train)
```

```
GridSearchCV(cv=None, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=Non
                                              criterion='entropy',
                                              max_depth=None, max_features=No
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=10,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [2, 4, 6, 8, 10, 12],
                         'min_samples_leaf': [1, 2],
                         'min_samples_split': [2, 3, 4]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

2) Fit model

```
model_id3 = gcv.best_estimator_
model_id3.fit(x_train,y_train)
y_train_pred = model_id3.predict(x_train)
y_test_pred = model_id3.predict(x_test)
```

3) Accuracy

```
print(f'Train score {accuracy_score(y_train_pred,y_train)}')
print(f'Test score {accuracy_score(y_test_pred,y_test)}')
```

```
Train score 0.775330396475771
Test score 0.75
```

4) Visualisation

```
dot_data = tree.export_graphviz(model_id3, out_file=None,
                      feature_names=feature_names,
                      class_names=class_names,
                      filled=True, rounded=True,
                      special_characters=True)
graph = graphviz.Source(dot_data)
graph
graph.render("ID3_With_pruning")
```

```
'ID3_With_pruning.pdf'
```

Analysis

After the experiments, we found out the following

> | Algorithms | Accuracy on Test Set (Without Pruning) | Accuracy on Test Set (With Pruning) |
> |---|---|---|
> | CART | 80% | 81.5% |
> | C4.5 | 100% | 100% |
> | ID3 | 85.5% | 82.5% |

Additionally, here is the comparision of the heights of the trees created by the algorithms

> | Algorithms | Height of Tree (Without Pruning) | Height of Tree (With Pruning) |
> |---|---|---|
> | CART | 11 | 5 |
> | C4.5 | 5 | 5 |
> | ID3 | 10 | 9 |