Python Tutorial 11

April 7, 2022

This tutorial is for Dr. Xin Tong's DSO 530 class at the University of Southern California in spring 2022. It aims to give you some supplementary code of Lecture 7 on how to implement *Decision Trees* and *Random Forest*.

0.1 Decision Trees

We only cover the classification tree in this tutorial. For regression trees, please check out the documentation of DecisionTreeRegressor at https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html.

[1]:
```
import numpy as np
import sklearn
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
print('The scikit-learn version is {}.'.format(sklearn.__version__))
```

```
The scikit-learn version is 0.23.2.
```

Note that you need to update `sklearn` to 0.22 or higher versions.

We will use `load_breast_cancer` to import the *Breast Cancer Wisconsin (Diagnostic) Data Set*. The breast cancer dataset is a classic and easy dataset for binary classification.

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe the characteristics of the cell nuclei present in the image.

[2]:
```
data = load_breast_cancer()
data.feature_names
```

[2]:
```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
```

```
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

[3]:
```
X, y = data.data, data.target
```

There are 569 instances in this dataset and each of them has 30 predictors.

[4]:
```
X.shape
```

[4]:
```
(569, 30)
```

y is a binary variable to represent two classes: *0* is *WDBC-Malignant* and *1* is *WDBC-Benign*.

[5]:
```
y[:40]
```

[5]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0])
```

We use `train_test_split` function to split the dataset into training data and test data.

[6]:
```
 X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=2)
```

First, we use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations. And we apply cost complexity pruning to this large tree to obtain a sequence of best subtrees, as a function of α.

We can use `min_samples_leaf` parameter to adjust the minimum number of observations required to be at a leaf node. The default value of `min_samples_leaf` is 1.

Please note that the parameter name `min_samples_leaf` involves the term "sample" when it actually refers to "observation". Although not perfect in statistics tradition, this usage is common in python packages.

[7]:
```
clf_tree = DecisionTreeClassifier(random_state=0)
# Note that although the tree building process looks like a deterministic␣
 ↪process, inside the package,
# there is some heuristic iterative algorithm used, so setting a random_state␣
 ↪will make sure of reproducibility.

path = clf_tree.cost_complexity_pruning_path(X_train, y_train)

ccp_alphas = path.ccp_alphas
```

`ccp_alphas` stores an array of alphas for pruning.

[8]:
```
print(ccp_alphas)
```

```
[0.         0.00232818 0.0068506  0.00730308 0.00985915 0.01533646
 0.02221077 0.02346023 0.02771098 0.33529903]
```

Then, we use K-fold cross-validation to choose α. Here, we use 10-fold CV to choose α on (X_train y_train) according to classification accuracy.

[9]:
```
from sklearn.model_selection import StratifiedKFold
kfolds = StratifiedKFold(n_splits = 10, shuffle = True, random_state = 1)# a␣
 ↪random␣state is set for reproducibility purpose

accuracies = []
for ccp_alpha in ccp_alphas:
    score_for_alpha = []
    for train_index, test_index in kfolds.split(X_train, y_train):
        clf = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
        clf.fit(X_train[train_index], y_train[train_index])
        y_pred = clf.predict(X_train[test_index])
        score = accuracy_score(y_pred, y_train[test_index])
        score_for_alpha.append(score)
    accuracies.append(sum(score_for_alpha)/len(score_for_alpha))
```

We can get the accuracies corresponding to the `ccp_alphas` and we can pick the best α to minimize the average classification error (i.e., maximize the accuracy).

[10]:
```
print("The accuracies: ",accuracies)
print("\nThe index corresponding to the maximum of the accuracies: ",np.
 ↪argmax(accuracies))
```

```
The accuracies:  [0.9368217054263566, 0.9368217054263566, 0.9461240310077519,
0.9461240310077519, 0.9507198228128461, 0.9411960132890366, 0.9153931339977852,
0.9153931339977852, 0.9154485049833887, 0.8597452934662237]

The index corresponding to the maximum of the accuracies:  4
```

At last, we use the selected α to retrain a tree on the entire (X_train, y_train) and evaluate on the (X_test, y_test).

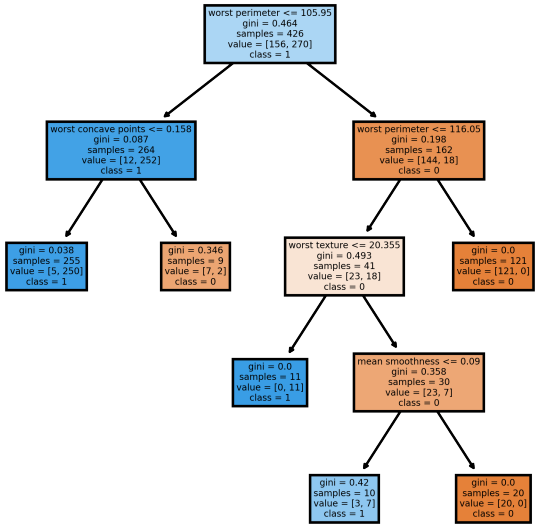
[11]:
```
# Use the selected alpha to retrain a tree on the entire (X_train,y_train)
alpha_cv = ccp_alphas[np.argmax(accuracies)]
clf_tree_final = DecisionTreeClassifier(random_state=0, ccp_alpha=alpha_cv)
clf_tree_final.fit(X_train, y_train)

# Evaluate on the (X_test,y_test)
y_pred_test = clf_tree_final.predict(X_test)
score_test = accuracy_score(y_test, y_pred_test)
print(score_test)
```

```
0.9440559440559441
```

[12]:
```
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=600)
tree.plot_tree(clf_tree_final,
               feature_names = data.feature_names,
```

```
                class_names=['0','1'],
                filled = True);
```



0.2 Random Forest

Again, we only cover the `RandomForestClassifier` in this tutorial. For `RandomForestRegressor`, please check out the documentation of RandomForestRegressor at https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html.

Note that `RandomForestRegressor` in `sklearn` has a default setting of `max_features = n_features`. This is not a commonly recommended option.

[13]:
```
from sklearn.ensemble import RandomForestClassifier
```

Parameters of `RandomForestClassifier`:

`n_estimators` (default 100) is the number of trees in the forest; `max_features` (default `sqrt(n_features)`) is the number of features to consider when looking for the best split.

[14]:
```
clf_rf = RandomForestClassifier(random_state=1, n_estimators = 200)
clf_rf.fit(X_train, y_train)

y_pred_rf = clf_rf.predict(X_test)
score_test_rf = accuracy_score(y_test, y_pred_rf)
print(score_test_rf)
```

```
0.951048951048951
```

[15]:
```
clf_rf.feature_importances_
```

[15]:
```
array([0.05869373, 0.01378831, 0.05026306, 0.04955652, 0.00550021,
       0.00785323, 0.05158817, 0.10049702, 0.00253205, 0.00430965,
       0.01767849, 0.00558484, 0.00955224, 0.03753636, 0.00228743,
       0.00376122, 0.00298014, 0.00521718, 0.00278189, 0.0036421 ,
       0.11841524, 0.01370842, 0.13218831, 0.11824133, 0.01192173,
       0.00820357, 0.02455718, 0.11803999, 0.01258987, 0.00653054])
```

[16]:
```
## plot the top 10 features in terms of Gini index
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=600)
sorted_idx = clf_rf.feature_importances_.argsort()[-10:]  ## remove [-10:] to␣
 ↪get a plot for all features
plt.barh(data.feature_names[sorted_idx], clf_rf.
 ↪feature_importances_[sorted_idx])
plt.xlabel("Random Forest Feature Importance")
```

[16]:
```
Text(0.5, 0, 'Random Forest Feature Importance')
```