Assignment 1: Decision Tree Grid Search

DTSC 680: Applied Machine Learning

Name: Juliana Meirelles

Directions

The main purpose of this assignment is for you to gain experience creating and visualizing a Decision Tree along with sweeping a problem's parameter space - in this case by performing a grid search. Doing so allows you to identify the optimal hyperparameter values to be used for training your model.

Preliminaries

Let's import some common packages:

In [29]:
```
import numpy as np
from sklearn import datasets
```

Load and Split Iris Data Set

Complete the following:

1. Load the `Iris` data set by calling the [load_iris() (https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html)](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html) function of the `datasets` library from `sklearn` - name the dictionary that is returned `iris`.
2. Call [train_test_split() (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html) with a `test_size` of 40% and a `random_state` of `0`. Save the output into `X_train`, `X_test`, `y_train`, and `y_test`, respectively. (Be sure to import the `train_test_split()` function first.)

In [30]:
```
from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

iris = load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.4,
```

Create a Single Decision Tree

Complete the following:

(Cell 1:)

1. Import the `DecisionTreeClassifier` class from the `sklearn.tree` library
2. Create a DecisionTreeClassifier object called `tree_clf` with a `random_state` of `42`
3. Fit the DecisionTreeClassifier object on the training data.

(Cell 2:)

4. Make a prediction on the test data, and name the predicted values output by the model `preds`.
5. Compute the performance of the model by measuring the accuracy score on the test set. You must import the [accuracy_score() (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html)](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html) function from the `sklearn.metrics` library. Name the accuracy score value you compute `acc_score`.
6. Print the accuracy score to the screen.

In [31]:
```
from sklearn.tree import DecisionTreeClassifier

tree_clf = DecisionTreeClassifier(random_state = 42)

tree_clf.fit(X_train, y_train)
```

Out[31]:
```
DecisionTreeClassifier(random_state=42)
```

In [32]:
```
from sklearn.metrics import accuracy_score

preds = tree_clf.predict(X_test)

acc_score = accuracy_score(y_test, preds)

print('Accuracy=%s' % (acc_score))
```

```
Accuracy=0.95
```

Perform Grid Search

Complete the following:

(Cell 1:)

1. Import the `GridSearchCV` class from the `sklearn.model_selection` library.
2. Create a dictionary called `param_grid` with three key-value pairs. The keys are `max_depth`, `max_leaf_nodes` and `min_samples_split`, and their respective values are `[1,2,3,4,5,8,16,32]`, `list(range(2, 20, 1))` and `[2,3,4,5,8,12,16,20]`.
3. Instantiate an object of the `GridSearchCV` class called `grid_search_cv`. Pass the following as input to the constructor:

- The model to be used. Use a `DecisionTreeClassifier` with a `random_state` parameter of `42`.
- The paramter grid.
- The hyperparameter `verbose=1`. (Look this up.)
- The number of cross-folds. Specify `cv=3`.

4. Call the `fit()` method to perform the grid search using 3-fold cross-validation.
5. Print the best parameters identified by the grid search using the `best_params_` attribute of the GridSearchCV object.

(Cell 2:)

6. Compute the predicted values `y_pred` using the test set `X_test`.
7. Calculate the accuracy, precision, and recall scores using the `accuracy_score()`, `precision_score()`, and `recall_score()` functions. Call these `acc_score`, `prec_score`, and `recall_score`, respectively. Set the average parameter to `micro` when calculating precision and recall to account for multiple classes.
8. Print all three scores to the screen.

In [33]:
```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'max_depth': [1,2,3,4,5,8,16,32],
    'max_leaf_nodes': list(range(2, 20, 1)),
    'min_samples_split': [2,3,4,5,8,12,16,20]
}

grid_cell_clf = DecisionTreeClassifier(random_state = 42)

grid_search_cv = GridSearchCV(grid_cell_clf, param_grid, verbose = 1, cv =

optimal_model = grid_search_cv.fit(X_train, y_train)

print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 1152 candidates, totalling 3456 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent w
orkers.

The best parameters are:  {'max_depth': 2, 'max_leaf_nodes': 3, 'min_samp
les_split': 2}

[Parallel(n_jobs=1)]: Done 3456 out of 3456 | elapsed:    3.8s finished
```

In [34]:
```
from sklearn.metrics import precision_score

from sklearn.metrics import recall_score

y_pred = optimal_model.predict(X_test)

acc_score = accuracy_score(y_test, y_pred)

prec_score = precision_score(y_test, y_pred, average = 'micro')

recall_score = recall_score(y_test, y_pred, average = 'micro')

print('Accuracy=%s' % (acc_score))
print('Precision=%s' % (prec_score))
print('Recall=%s' % (recall_score))
```

```
Accuracy=0.8666666666666667
Precision=0.8666666666666667
Recall=0.8666666666666667
```

Visualize Optimal Decision Tree as Text

Instantiate a new `DecisionTreeClassifier` object, and use the `best_params_` attribute of the `grid_search_cv` object to specify the best `max_depth`, `max_leaf_nodes` and `min_samples_split` values calculated from the grid search along with a `random_state` of `42`. Retrain the "optimal" (for the few parameters that we swept) decision tree.

Next, use the [tree.export_text() (https://scikit-learn.org/stable/modules/generated/sklearn.tree.export_text.html)](https://scikit-learn.org/stable/modules/generated/sklearn.tree.export_text.html) method to visualize the "optimal" decision tree. This function takes a trained classifier as its first parameter, and a set of feature names as its second parameter (the feature names are included in the `iris` dictionary returned from the `load_iris()` function). The result is a text based visualization of the decision tree. Note that this method returns a string, so you'll want to `print()` the result to get it to look right.

In [35]:
```
from sklearn.tree import export_text

best_tree_clf = DecisionTreeClassifier(max_depth = 2, max_leaf_nodes = 3, m

best_tree_clf.fit(X_train, y_train)

tree_as_text = export_text(best_tree_clf, feature_names= (['sepal length (c
                                                          'sepal width (cm)'
                                                          'petal length (cm)
                                                          'petal width (cm)'


print(tree_as_text)
```

```
|--- petal length (cm) <= 2.35
|   |--- class: 0
|--- petal length (cm) >  2.35
|   |--- petal length (cm) <= 5.05
|   |   |--- class: 1
|   |--- petal length (cm) >  5.05
|   |   |--- class: 2
```

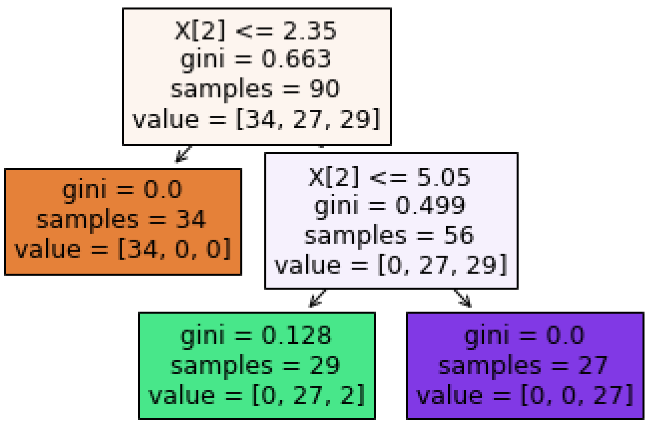
Visualize Optimal Decision Tree as Image

Use the `tree.plot_tree()` method to visualize the "optimal" decision tree, which takes a trained classifier as its only parameter and returns a graphical visualization of the decision tree. Use `filled=True` as an argument to the method to add color to the image.

In [36]:
```python
from sklearn import tree

tree.plot_tree(best_tree_clf, filled = True)
```

Out[36]:
```
[Text(133.92000000000002, 181.2, 'X[2] <= 2.35\ngini = 0.663\nsamples = 9
0\nvalue = [34, 27, 29]'),
 Text(66.96000000000001, 108.72, 'gini = 0.0\nsamples = 34\nvalue = [34,
0, 0]'),
 Text(200.88000000000002, 108.72, 'X[2] <= 5.05\ngini = 0.499\nsamples =
56\nvalue = [0, 27, 29]'),
 Text(133.92000000000002, 36.23999999999998, 'gini = 0.128\nsamples = 29
\nvalue = [0, 27, 2]'),
 Text(267.84000000000003, 36.23999999999998, 'gini = 0.0\nsamples = 27\nv
alue = [0, 0, 27]')]
```



Critical Analysis

In your own words, describe or interpret the role of the gini score criterion in the decision tree algorithm. How does this compare to the entropy impurity measure? Finally, sklearn uses the CART (Classification and Regression Tree) algorithm to train Decision Trees. How does this algorithm determine the feature and threshold value to use for splitting at each step of the Decision Tree algorithm? It may be helpful to look at outside resources to help you answer these questions (The YouTube channel ["StatQuest" (https://youtu.be/7VeUPuFGJHk)](https://youtu.be/7VeUPuFGJHk) has some excellent videos on Decision Trees for those of you that like visual explanations.)

Make sure that you answer all the questions above. I am looking for **meaningful content** here that **goes into detail**. Don't just copy from the textbook or rush through answering this question.

Answer:

The role of the gini score criterion in the decision tree algorithm is as of a measure that describes the homogeneity of a class in a data set. Each node in a decision tree, or classification grouping, contains a class. The class is the main type of grouping included on a node - the one we are aiming to describe. A gini score of 0 within one node means that all of the instances or examples in the node belong to the same class. For our iris data set, a gini score of 0 could be that all the flowers in one node are of the setosa class.

The gini score criterion is comparable to the entropy impurity measure because they both aim to represent the amount to which a node's instances have indentical or non-identical classes. Similarly to a gini score of 0, an entropy value of 0 means that a node contains only one type of class. A key difference between the two is computation time (with the Gini score being faster, which is likely why it is popular). Additionally, entropy creates more balanced trees than the Gini score. The StatQuest video mentions that there are many ways to measure impurity, so the Gini impurity and entropy are just two options with their own pros and cons.

The CART algorithm determines the feature and threshold value to use for splitting at each step of the Decision Tree algorithm by dividing the training set into a feature and a threshold. The algorithm choose a pairing of feature and threshold by selecting groupings with the lowest impurity. The algorithm continues to split these sets in subsets and continues splitting until it reaches a maximum, which we call the maximum depth. The maximum depth is set using a hyperparameter (example: max_depth = 2). Alternatively, it will stop when it cannot find another split in the set that produces "pure" subsets, meaning the data will be increasingly impure if splitting continues.

Ungraded Critical Thinking Question

Compare the accuracy score from the first Decision Tree to the accuracy score after you performed the grid search. How does it differ? It is most likely that you will find the accuracy score has decreased. Is that what you had expected? We perform a round of grid searching in order to elucidate the optimal hyperparameter values. Why, then, has the accuracy score decreased? Most importantly, what caused this decrease in the accuracy score and why? Explain your answer.

Answer:

My accuracy score for the first Decision tree was 95% and my accuracy score for the grid search model was around 87%, so my accuracy score decreased when performing a grid search to include optimal hyperparameters. The accuracy score has probably decreased because the model is overfitting to the data. The hyperparameters cause the data to be optimized for the specific data that falls under those hyperparameters, but therefore will not generalize well for other data points that fall outside of the hyperparameters. This outcome is not what I had expected because in my head I thought that adding these hyperparameters would make the model as accurate as possible because it would be understanding the data more closely, but I think this is an interesting concept to try and understand because hyperparameters are used as controls and could become too narrow.

In [ ]:

```

```