Data Science and Database Technologies

HOMEWORK #2 - Revalor Riccardo - s339423

Preliminary Step: Dataset Creation, Cleaning and Encoding
After I've loaded the content of the xlxs file into the pandas Dataframe, I clean the data. Since all of the data in the original dataframe are categorical, I have to apply an Encoding tecnique in order to map them to integers. Since Label encoding tends to impose an arbitrary order on categorical data, which can be misleading, I prefer to use **One Hot Encoding** to treat the data. Before doing it, I apply a manual **label binarization** to explicitly convert binary categorical variables (e.g., 'no'/'yes', or 'left'/'right') into numerical values (0 and 1). This approach is used for just four columns in the dataset (*node-caps, breast, irradiat, Class*) and significantly reduces dataset dimensionality compared to OneHotEncoding, which would create separate columns for each category, simplifying the model and saving memory.

```
def clean(dataframe):
    dataframe = dataframe.dropna()
    dataframe = dataframe[~dataframe.apply(lambda row: '?' in row.values, axis=1)]
    dataframe["node-caps"] = dataframe["node-caps"].replace({"'no'": 0, "'yes'": 1})
    dataframe["breast"] = dataframe["breast"].replace({"'right'": 0, "'left'": 1})
    dataframe["irradiat"] = dataframe["irradiat"].replace({"'no'": 0, "'yes'": 1})
    dataframe["Class"] = dataframe["Class"].replace({"'no-recurrence-events'": 0, "'recurrence-events'": 1})
    return dataframe
```

Then I apply One Hot Encoding to encode all the other columns:

```
#one hot encoding, use get dummies of scikit learn
dataframe = pd.get_dummies(dataframe, columns= ['age', 'menopause', 'tumor-size', 'inv-nodes', 'deg-malig', 'breast-quad'])
dataframe = dataframe.replace({False: 0, True: 1})
```

The target and the features are selected this way:

```
X = dataframe.copy().drop("Class", axis=1)  # All columns except the last X = features
y = dataframe["Class"]   # Class column (last column) y = target
```

Question N.1 - Decision Tree classified with parameter *max_depth* manually set to 5

1) I identify the most discriminative attribute for class prediction this way:

```
# Identify the most discriminative attribute
feature_importances = clf.feature_importances_
most_discriminative = X.columns[feature_importances.argmax()]
print(f"Most discriminative attribute: {most_discriminative}")
```

✓ 0.0s

```
Most discriminative attribute: inv-nodes_'0-2'
```

It's *inv-nodes_'0-2'* attribute. By looking at the Tree I see that attributes *deg_malig_'3'* and *tumor-size_'10-14'* are also quite significative.

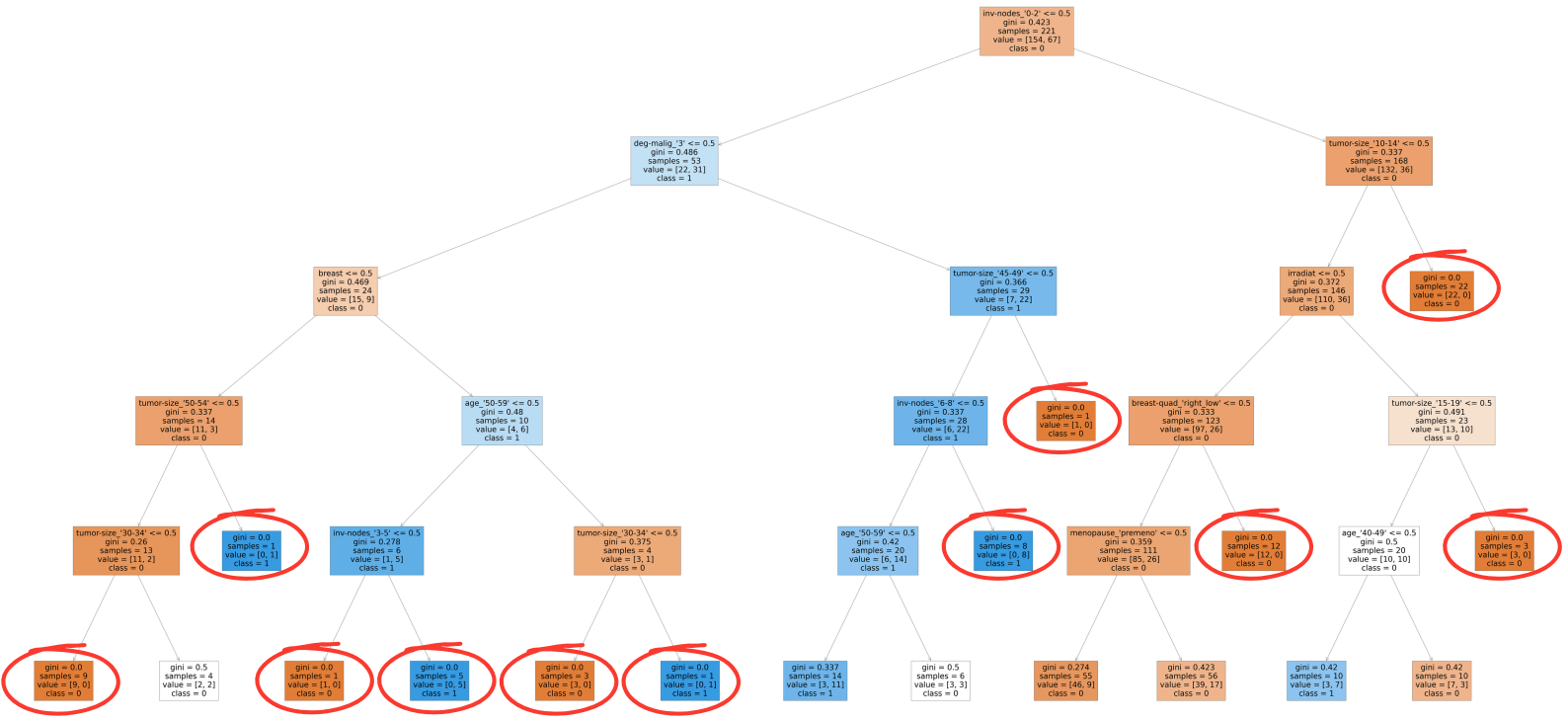
The height of the generated Decision Tree is 5:

```
# Determine the height of the tree
tree_height = clf.get_depth()
print(f"Height of the Decision Tree: {tree_height}")
```

✓ 0.0s

```
Height of the Decision Tree: 5
```

This characteristics can be observed also by looking at the generated Decision Tree, drawn with matplotlib and sklearn.tree. I've circled all the pure partitions, characterized by a null Gini index.



In this case, the accuracy and Classification Report are:

```
print(classification_report(y_test, predictions))
print(accuracy_score(y_test, predictions))
```

✓ 0.0s

```
              precision    recall  f1-score   support

           0       0.76      0.90      0.83        42
           1       0.33      0.14      0.20        14

    accuracy                           0.71        56
   macro avg       0.55      0.52      0.51        56
weighted avg       0.65      0.71      0.67        56

0.7142857142857143
```

Question N.2 - Influence of Decision Tree parameters

Here I've created a function to play with the different parameters of the Decision Tree Classifier and see their effects on the overall Tree if changed:

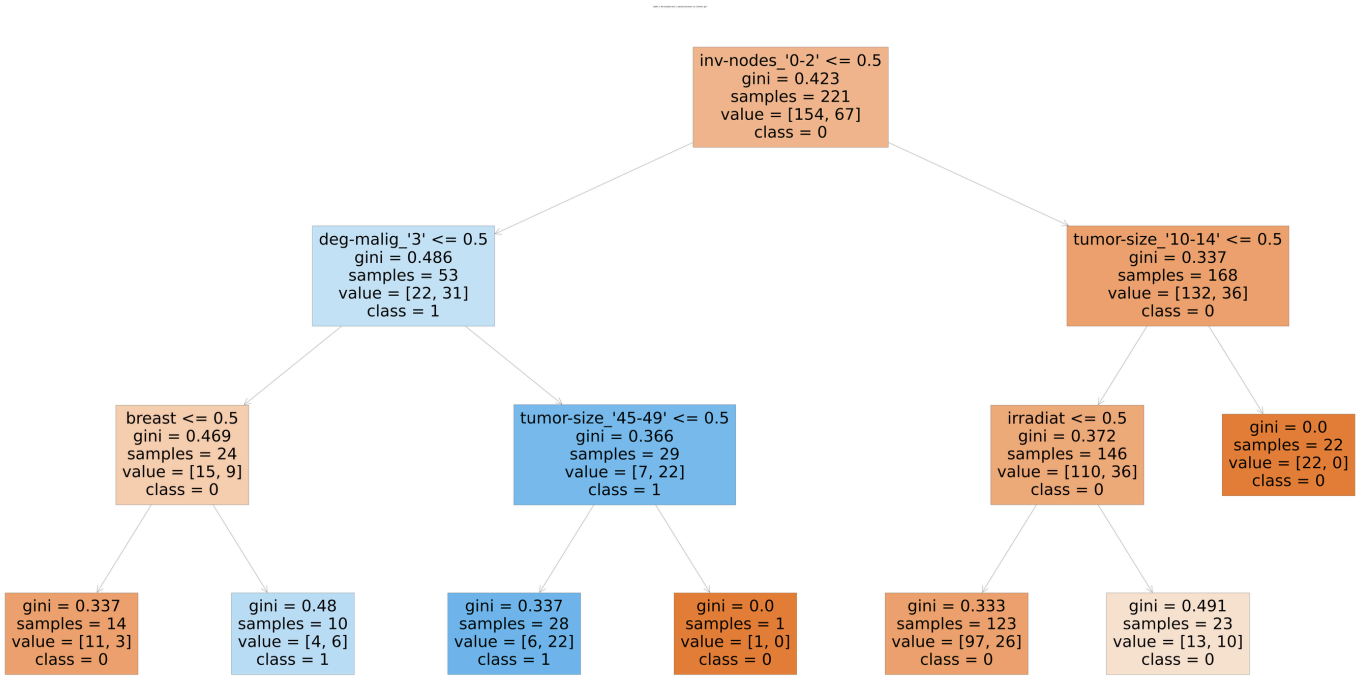
```
#Function used to show the impact of different hyperparameters on the tree
def train_and_plot(max_depth=None, min_samples_leaf=1, min_impurity_decrease=0.0, criterion='gini'):
    clf = DecisionTreeClassifier(
        criterion=criterion,
        max_depth=max_depth,
        min_samples_leaf=min_samples_leaf,
        min_impurity_decrease=min_impurity_decrease,
        random_state=42
    )
    clf.fit(X_train, y_train)

    # Plot the tree
    plt.figure(figsize=(200, 100))
    tree.plot_tree(clf, feature_names=X.columns, class_names=clf.classes_.astype(str), filled=True)
    plt.title(f"Depth: {max_depth}, Min Samples Leaf: {min_samples_leaf}, Impurity Decrease: {min_impurity_decrease}, Criterion: {criterion}")
    plt.show()
```

The **default** parameters of the constructor are:

- *max_depth = None* (no threshold on Tree height)
- *min_samples_leaf = 1*
- *min_impurity_decrease = 0.0*
- *criterion='gini'*

First configuration: max_depth=3, all the other params are set to default values



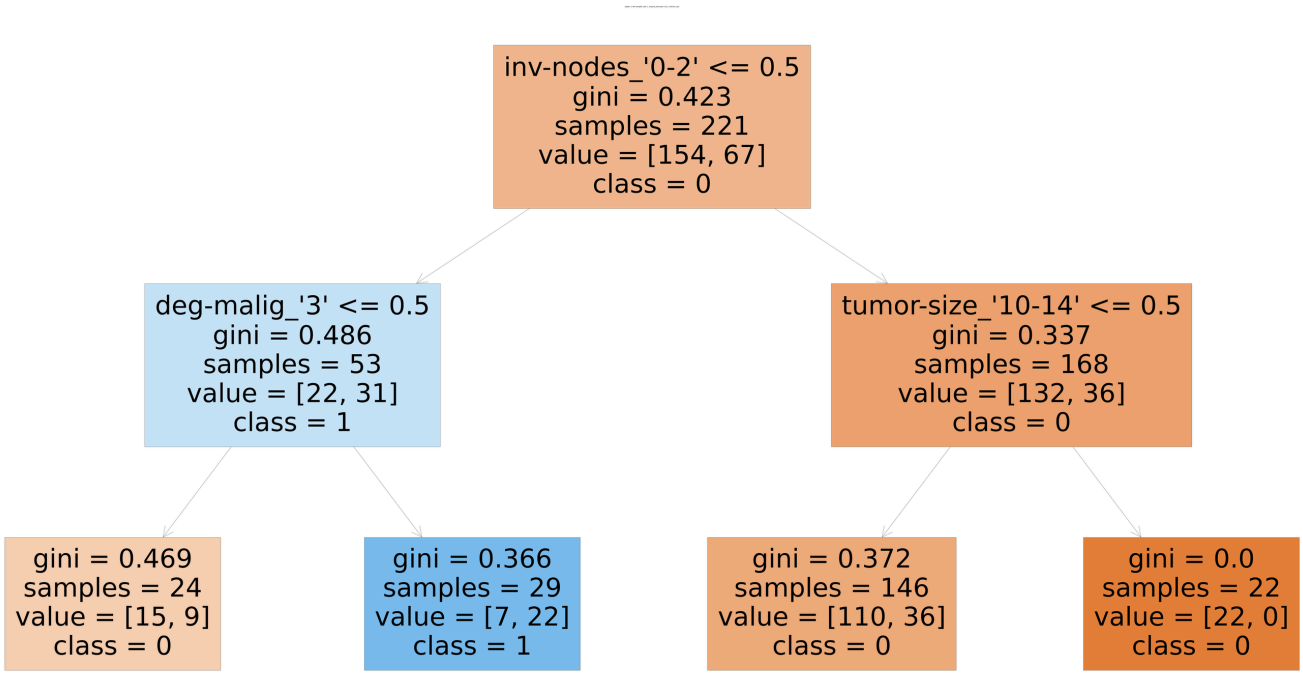
Classification Report:

	precision	recall	f1-score	support
0	0.81	0.93	0.87	42
1	0.62	0.36	0.45	14
accuracy			0.79	56
macro avg	0.72	0.64	0.66	56
weighted avg	0.77	0.79	0.76	56

In comparison with the Tree generated setting max_depth equal to five, overall Accuracy improves (79% vs 71%) and also the Recall and F1 score for the minority Class (class 0) are improved, meaning now the model handles the it slightly better. I think that, because of the very **low number of rows** in the datasets (277 after the cleaning process), setting a lower max_depth actually simplifies the model, helping it generalize better. In other words, in this case a deeper Tree risks **overfitting**.

Second configuration: max_depth=5, min_imminpurity_decrease=0.01, all the other params are set to default values

Generated Tree and Classification Report:

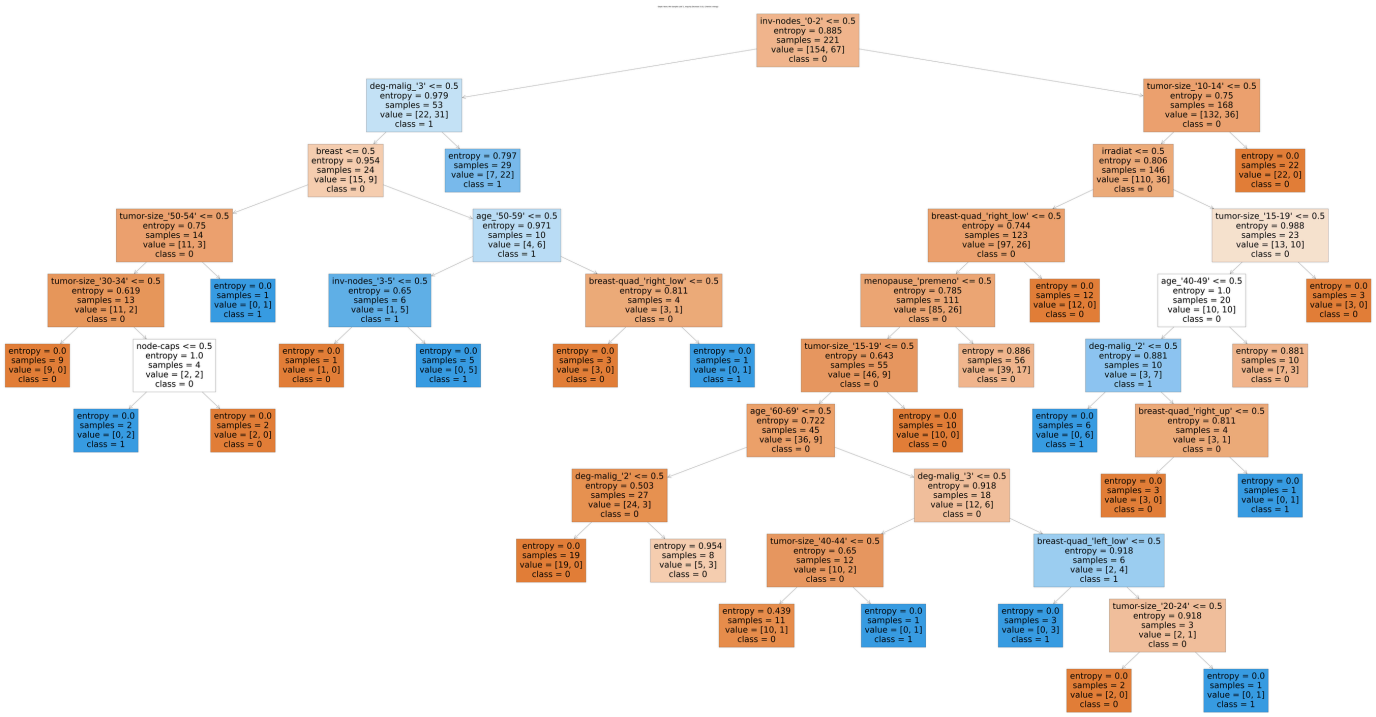


	precision	recall	f1-score	support
0	0.82	0.98	0.89	42
1	0.83	0.36	0.50	14
accuracy			0.82	56
macro avg	0.83	0.67	0.70	56
weighted avg	0.82	0.82	0.79	56

The F1 Score for Class 1 improved to 0.50, suggesting a better balance between the two classes. Recall was the same as the previous configuration for Class 1, and improved by 5 pp for Class 0, for which is very high. Precision improved for both classes and overall Accuracy reached 82%. I used this configuration because I wanted to see if I would reduce complexity and prevent overfitting, and in facts, althought max_depth was manually set to 5, the height of the Tree here is 3, not 5. This means that setting a **slightly** higher value of **min_purity_decrease** improves **model generalization**.

Third configuration: min_impurity_decrease=0.01, criterion='entropy', all the other params are set to default values

Generated Tree and Classification Report:

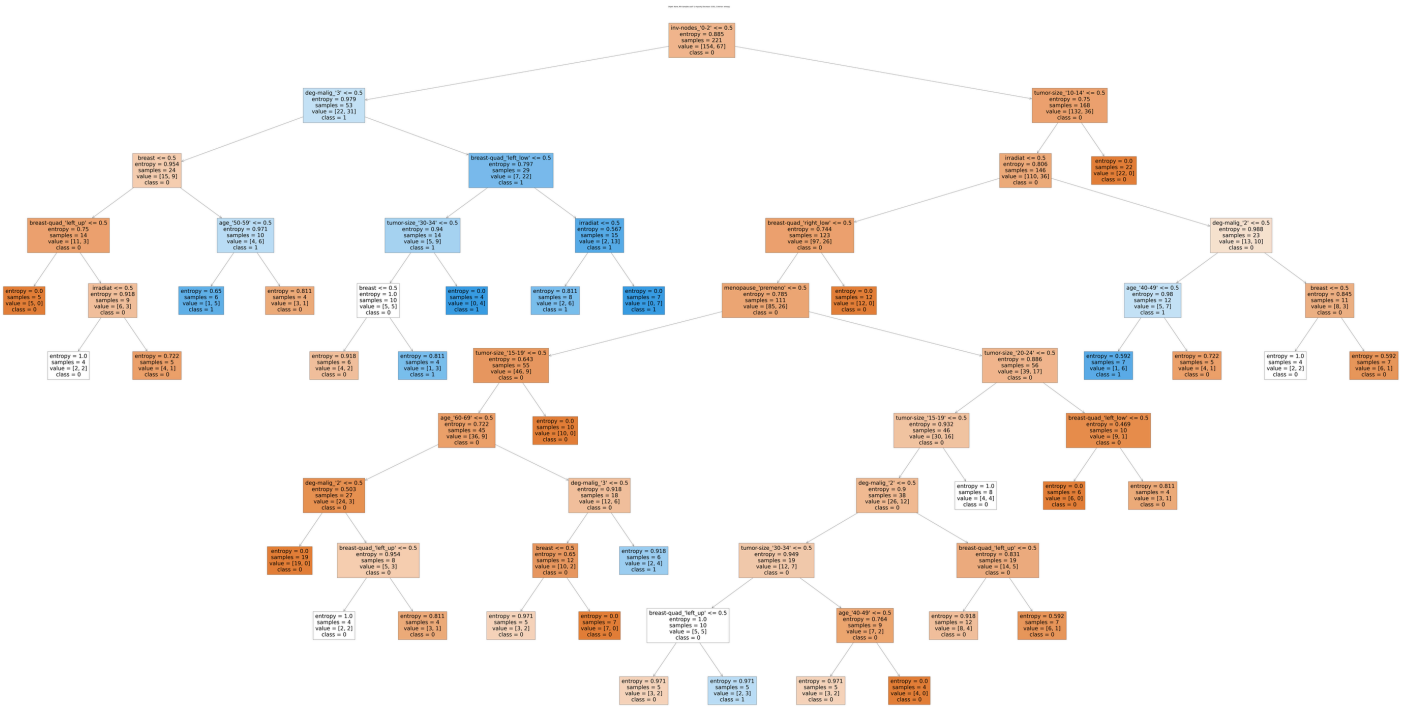


	precision	recall	f1-score	support
0	0.82	0.95	0.88	42
1	0.71	0.36	0.48	14
accuracy			0.80	56
macro avg	0.77	0.65	0.68	56
weighted avg	0.79	0.80	0.78	56

In this case I do not set max_depth, so the model is “free” to create as many levels of the Tree as it needs. Also, using the Entropy criterion instead of Gini index does not limit the actual height to 3 as the previous case. I note that the absence of a limitation regarding Tree depth lets the model be more complex, but this **does not result in better performance! Instead, the overall accuracy decreases by 2 pp.**

Fourth configuration: min_samples_leaf=4, min_impurity_decrease=0.001, criterion='entropy', all the other params are set to default values

Generated Tree and Classification Report:

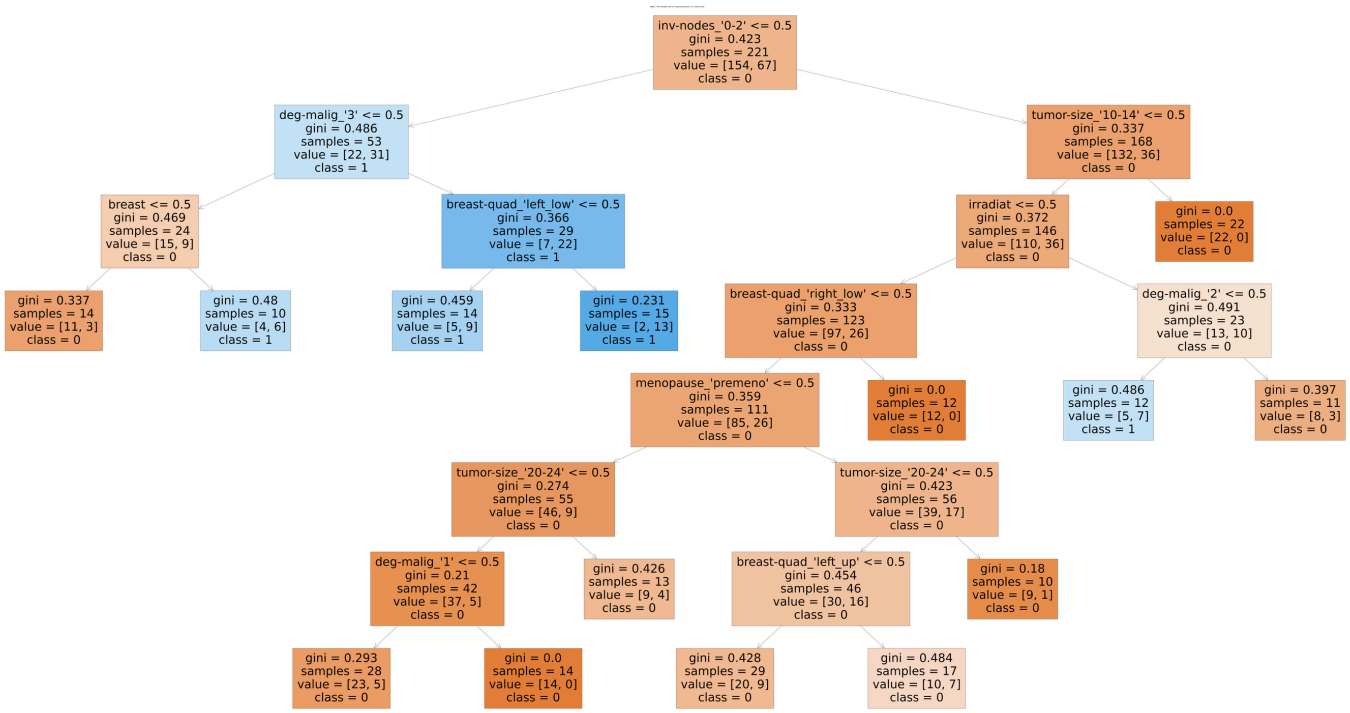


	precision	recall	f1-score	support
0	0.82	0.98	0.89	42
1	0.83	0.36	0.50	14
accuracy			0.82	56
macro avg	0.83	0.67	0.70	56
weighted avg	0.82	0.82	0.79	56

Here the precision for the Class 1 is improved. I think that setting a slightly higher number of min_samples_leaf (whose default value is 1) actually imprpoves genralizations and **reduces overfitting**.

Fifth configuration: max_depth=7, min_samples_leaf=10, criterion='gini', all the other params are set to default values

Generated Tree and Classification Report:



	precision	recall	f1-score	support
0	0.80	0.88	0.84	42
1	0.50	0.36	0.42	14
accuracy			0.75	56
macro avg	0.65	0.62	0.63	56
weighted avg	0.73	0.75	0.73	56

In this case I think that higher number of min_samples_leaf may cause **too much generalization** and potentially **underfitting**: the model is forced to make broader, more generalized splits. The tree will avoid creating very specific or narrow splits that would capture small variations in the data.

Summary:

- max_depth: a lower value of it reduces overfitting (although a value that is too low, ex. 1 or 2, causes underfitting); a higher value of it may cause overfitting becase the mode will be too sensitive to the training data.
- min_impurity_decrease: larger values (>> 0) reduce splits, leading to underfitting. The defaut and lowest value is obviously 0, which allows the tree to split as much as possible.
- min_samples_leaf: larger values force the model to create bigger leaves and may prevent overfitting. Values that are too high of course introduce too much generalization and may lead to underfitting.
- criterion: Gini Index tends to lead to larger, more balanced splits whereas Entropy Criterion is generally more sensitive to imbalances (and is also more expensive in terms of complexity than just applying the Gini Index).