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
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'
```

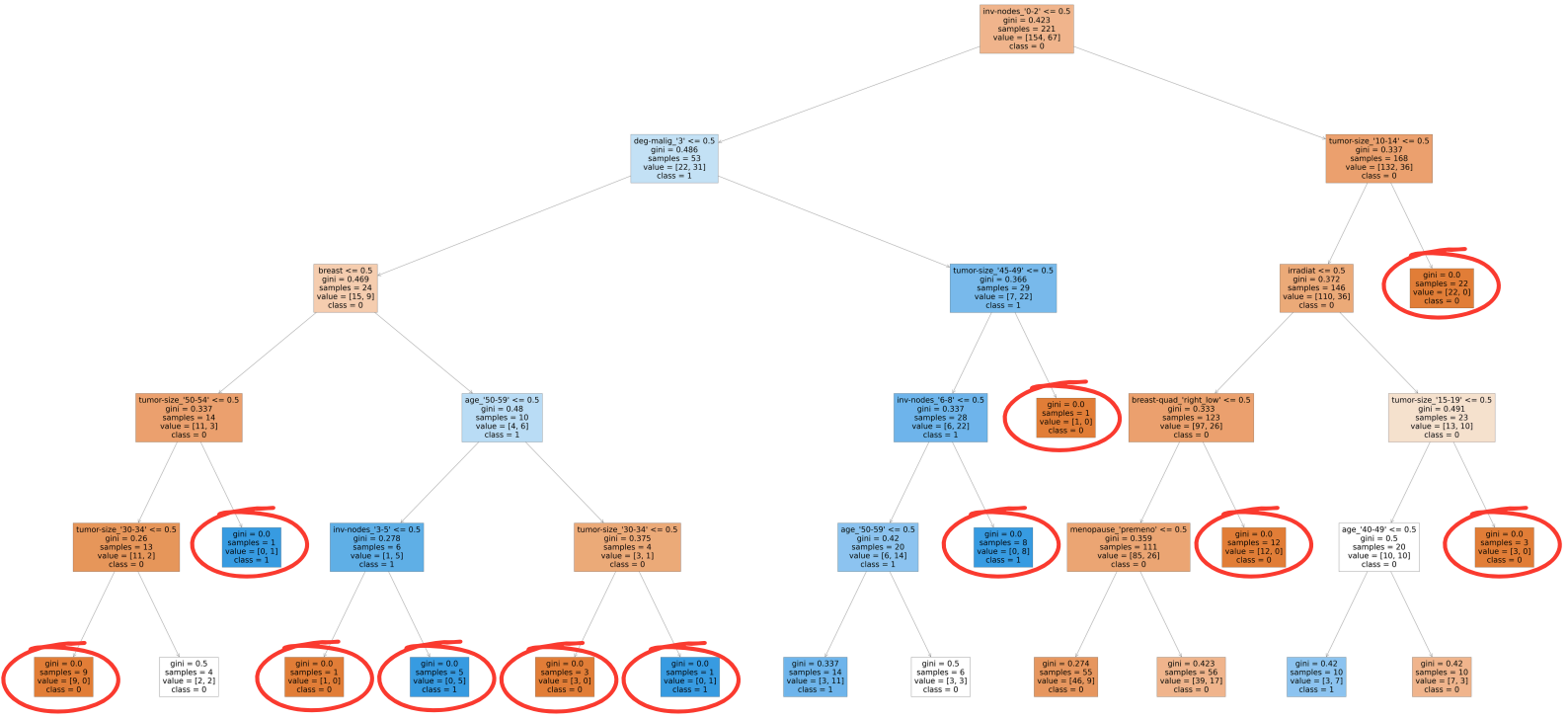
It's *inv-nodes_'0-2'* attribute. By looking at the Tree I see that attributes *deg_malig_'3'* and *tumor-size_'10-14'* are also quite significative.
The height of the generated Decision Tree is 5:

```
# 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, 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:

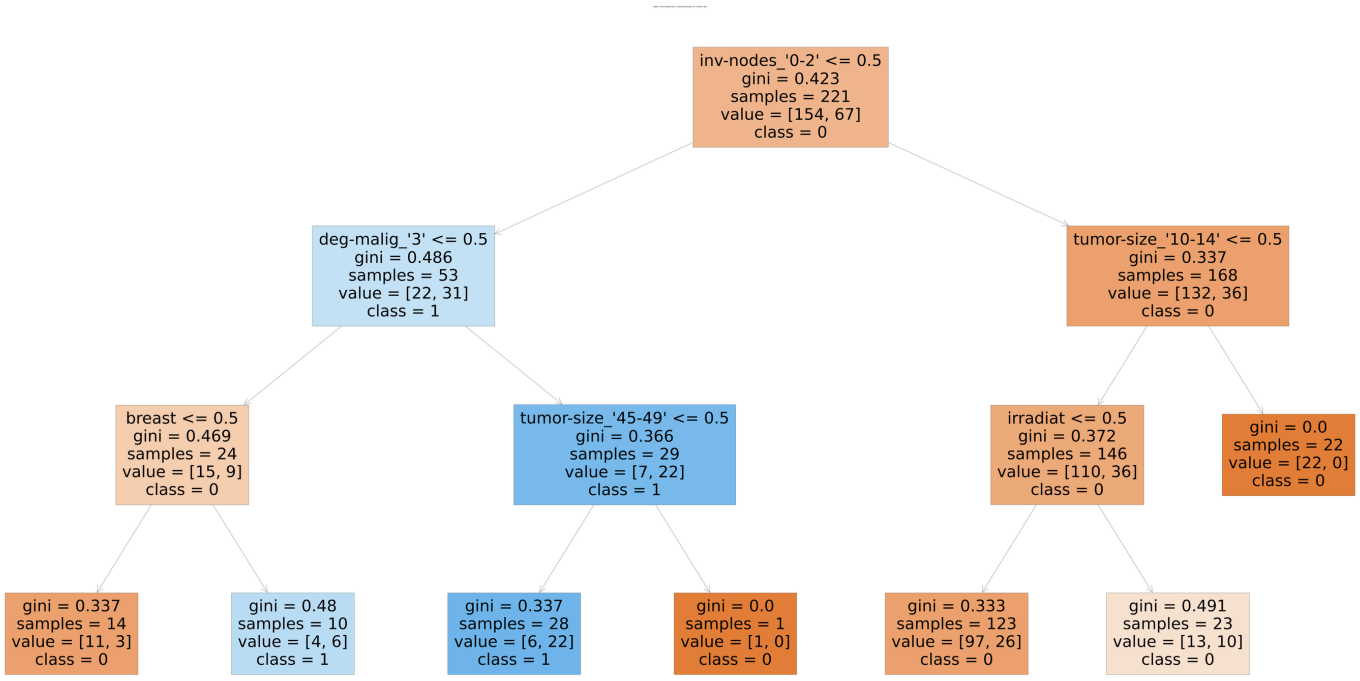
```
#Function used to show the impact of different hyperparameters on the tree
def train_and_plot(max_depth=None, min_samples_leaf=1, min_impurity_decrease=0.0, criterion='gini'):
    clf = DecisionTreeClassifier(
        criterion=criterion,
        max_depth=max_depth,
        min_samples_leaf=min_samples_leaf,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

Generated Tree and Classication Report:

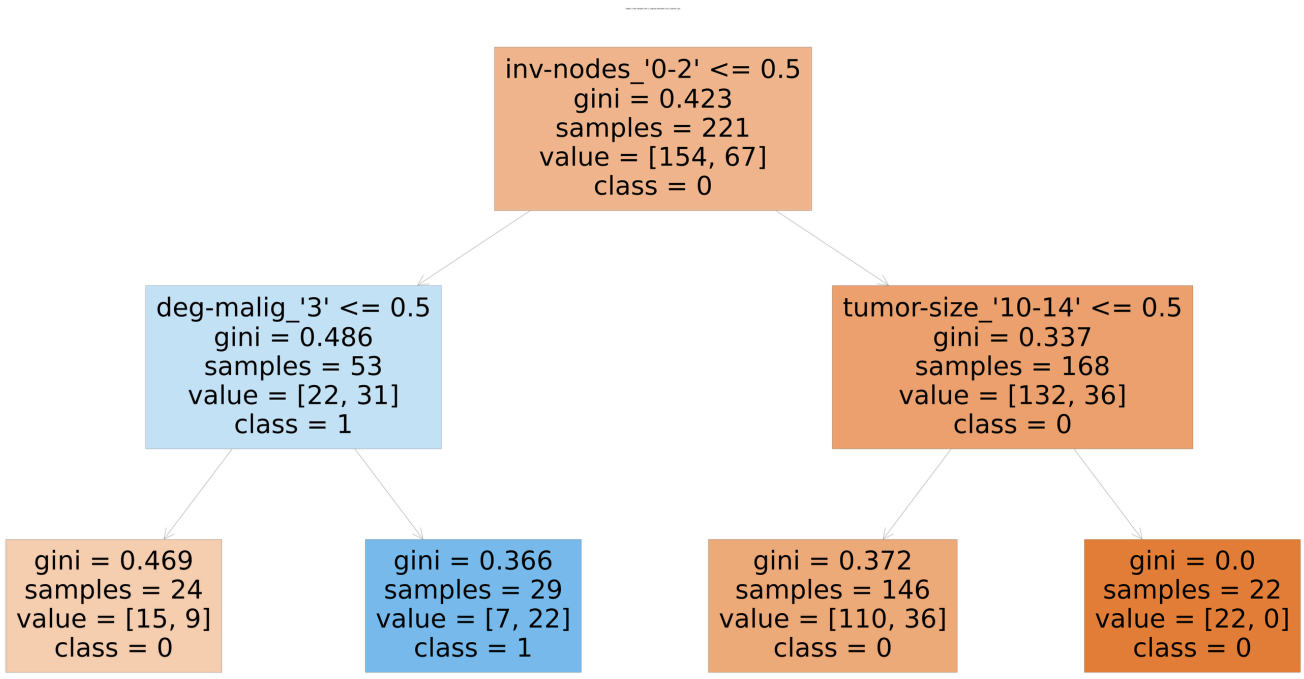


	precision	recall	f1-score	support
0	0.81	0.93	0.87	42
1	0.62	0.36	0.45	14
accuracy			0.79	56
macro avg	0.72	0.64	0.66	56
weighted avg	0.77	0.79	0.76	56

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:

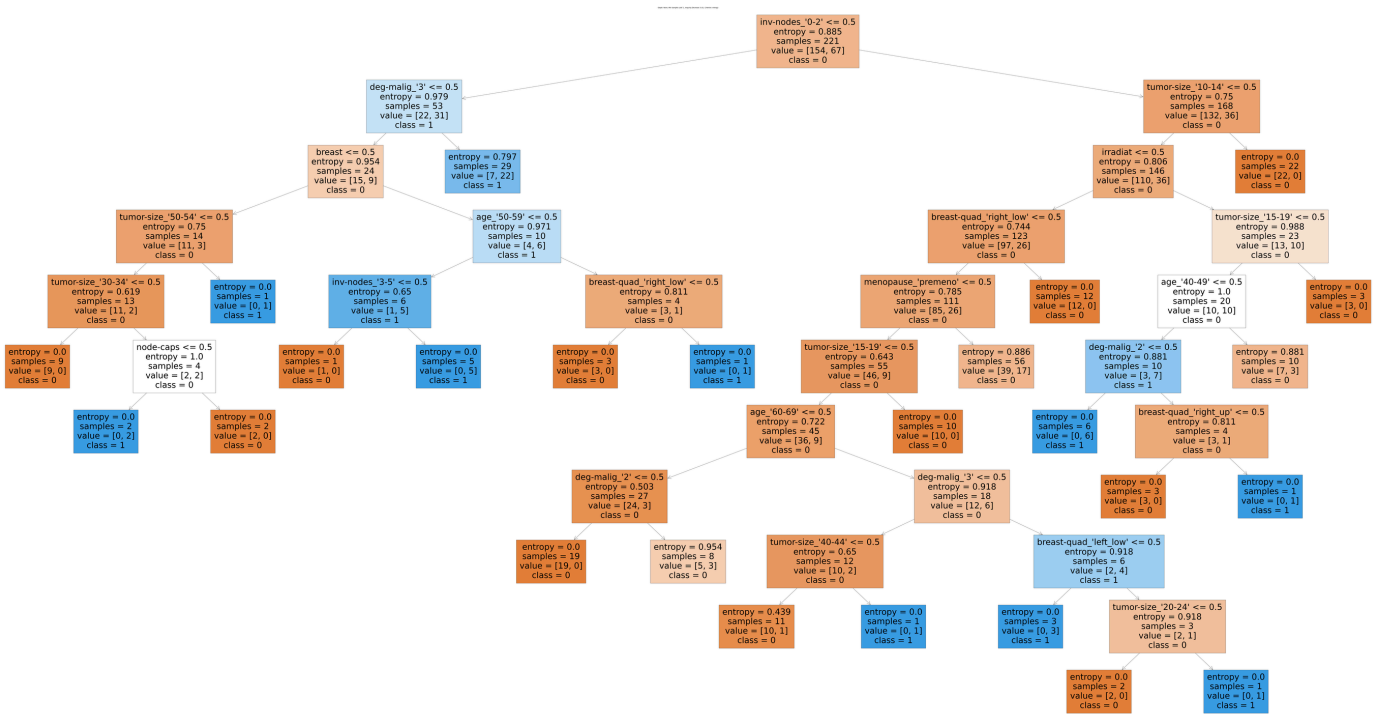


	precision	recall	f1-score	support
0	0.82	0.98	0.89	42
1	0.83	0.36	0.50	14
accuracy			0.82	56
macro avg	0.83	0.67	0.70	56
weighted avg	0.82	0.82	0.79	56

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

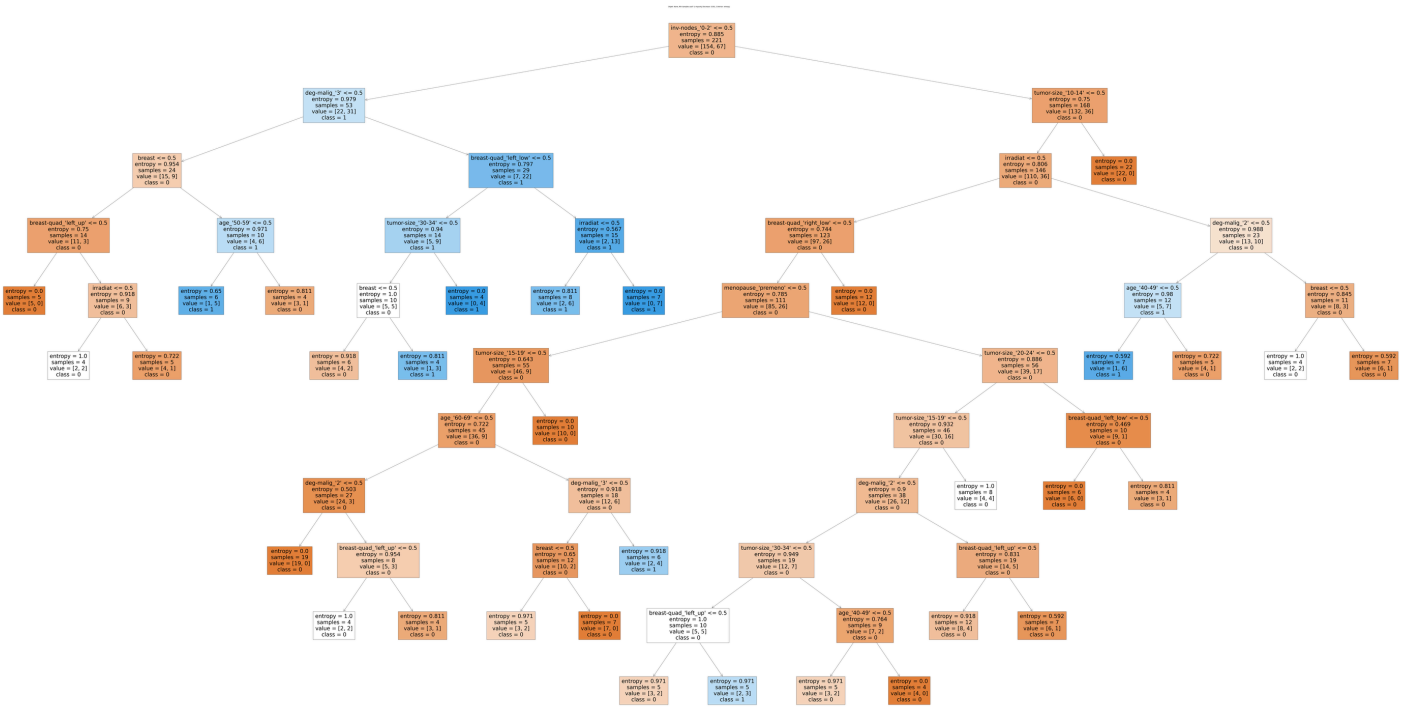
Generated Tree and Classification Report:



	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.**

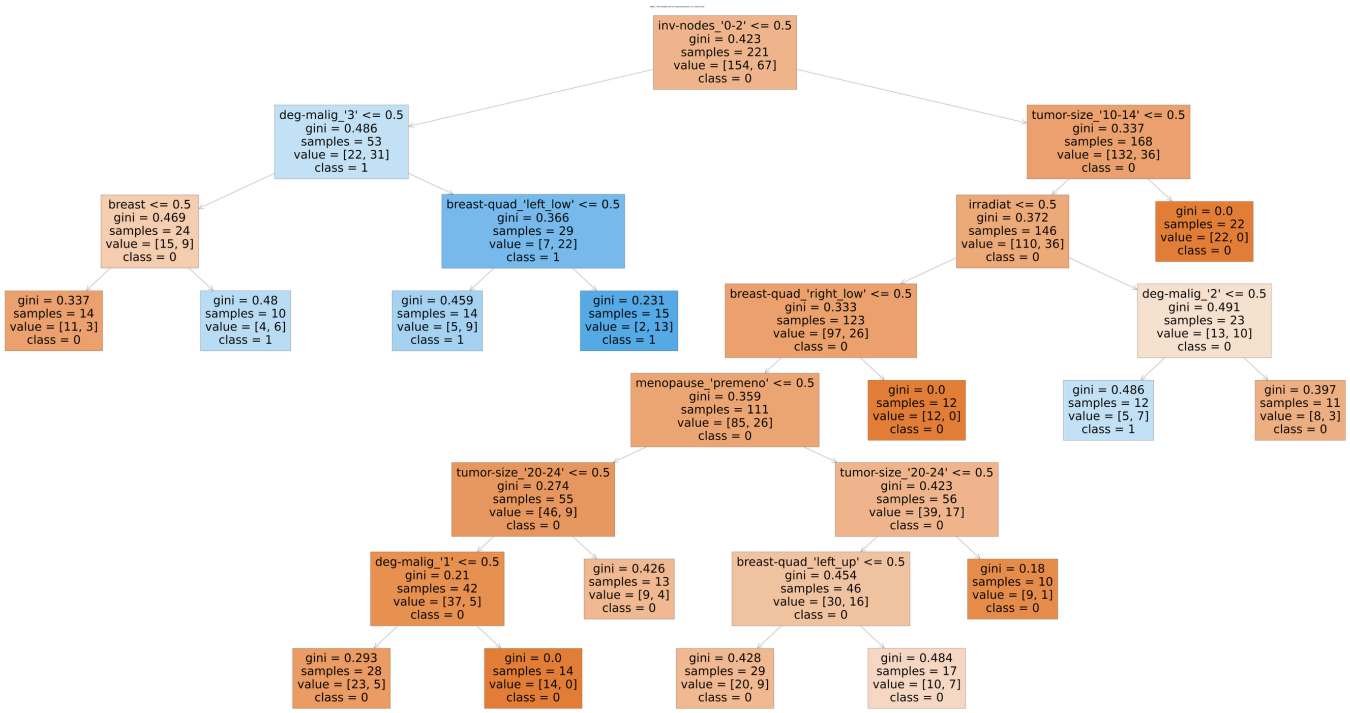
Fourth configuration: min_samples_leaf=4, min_impurity_decrease=0.001, criterion='entropy', all the other params are set to default values
Generated Tree and Classification Report:



	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).

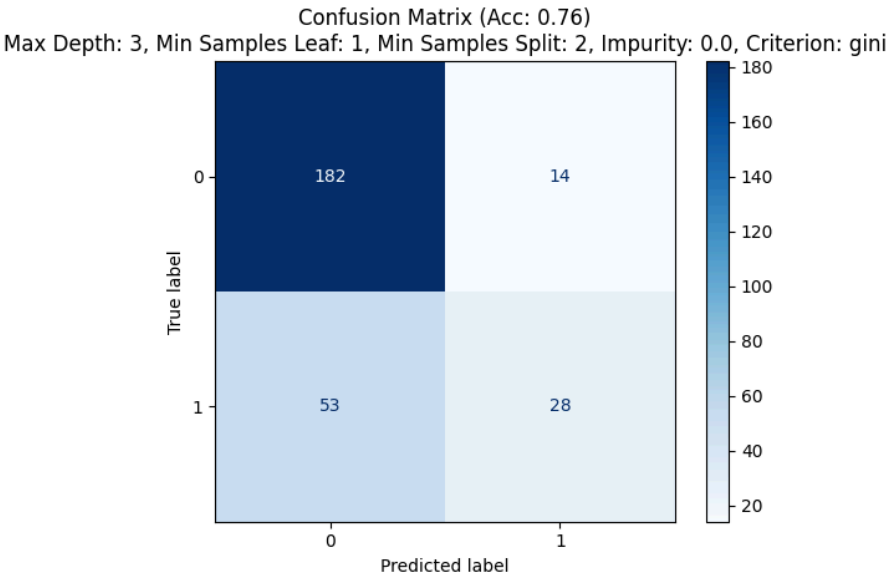
Question N.3 - 10-fold Stratified Cross-Validation

In order to exeperiment with parameters I have created the following functions that shows the Confusion Matrix using seaborn library, the generated Tree and returns the Classification Report:

```python
def experiment_with_cv(X, y, max_depth=None, min_samples_leaf=1,min_samples_split=2, 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,
        min_samples_split=min_samples_split,
        random_state=42
    )
    clf.fit(X_train, y_train)

    #Stratified k fold with k = 10
    stk10 = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
    predictions = cross_val_predict(clf, X, y, cv=stk10)

    #create confusion matrix
    confMatrix = confusion_matrix(y, predictions)
    print(confMatrix)

    #display confMatrix
    accuracy = accuracy_score(y, predictions)
    ConfusionMatrixDisplay(confusion_matrix=confMatrix, display_labels=np.unique(y)).plot(cmap='Blues')
    plt.title(f"Confusion Matrix (Acc: {accuracy:.2f})\nMax Depth: {max_depth}, Min Samples Leaf: {min_samples_leaf}, Min Samples Split: {min_samples_split}, Impurity: {m
    plt.show()

    # 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}, Min Samples Split: {min_samples_split}, Impurity Decrease: {min_impurity_decrease}, Criterion: {
    plt.show()

    return classification_report(y, predictions)
```

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

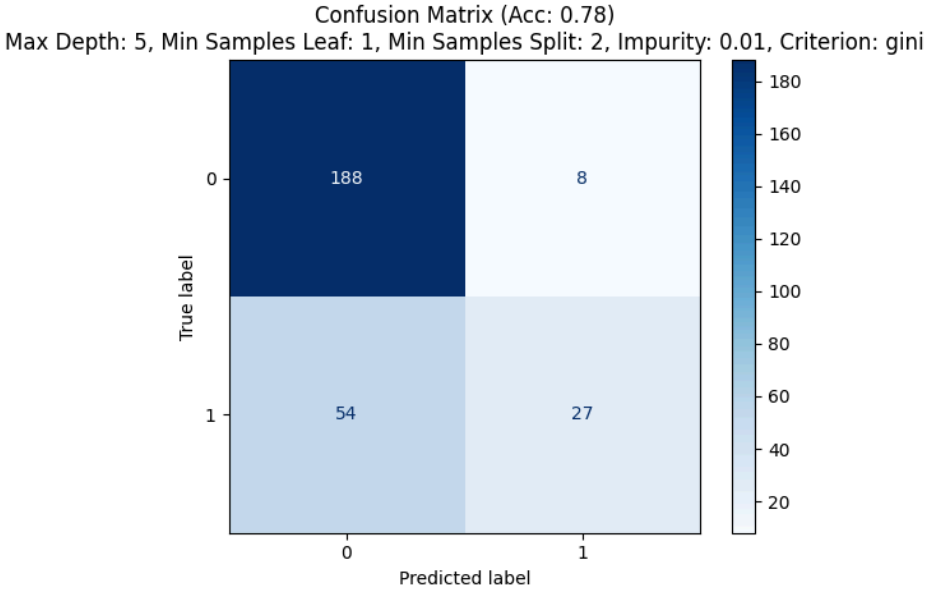
Confusion Matrix and Classification Report:



	precision	recall	f1-score	support
0	0.77	0.93	0.84	196
1	0.67	0.35	0.46	81
accuracy			0.76	277
macro avg	0.72	0.64	0.65	277
weighted avg	0.74	0.76	0.73	277

The model performs well on Class 0 but fails to predict class 1 in many cases (53 over 81). This is because the dataset is very **imbalanced towards Class 0!**

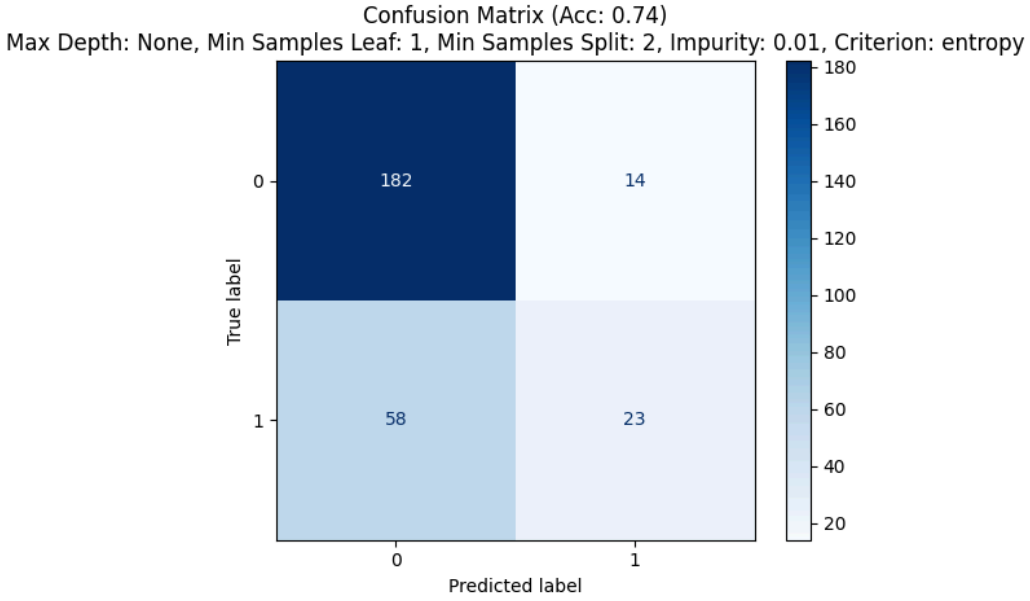
Second configuration: max_depth=5, min_imminpurity_decrease=0.01, all the other params are set to default values
Confusion Matrix and Classification Report:



	precision	recall	f1-score	support
0	0.78	0.96	0.86	196
1	0.77	0.33	0.47	81
accuracy			0.78	277
macro avg	0.77	0.65	0.66	277
weighted avg	0.78	0.78	0.74	277

Here I can say the True Class 0 values (188) become slightly better, although the False Class 0 values (54) remain very numerous and this is not optimal.

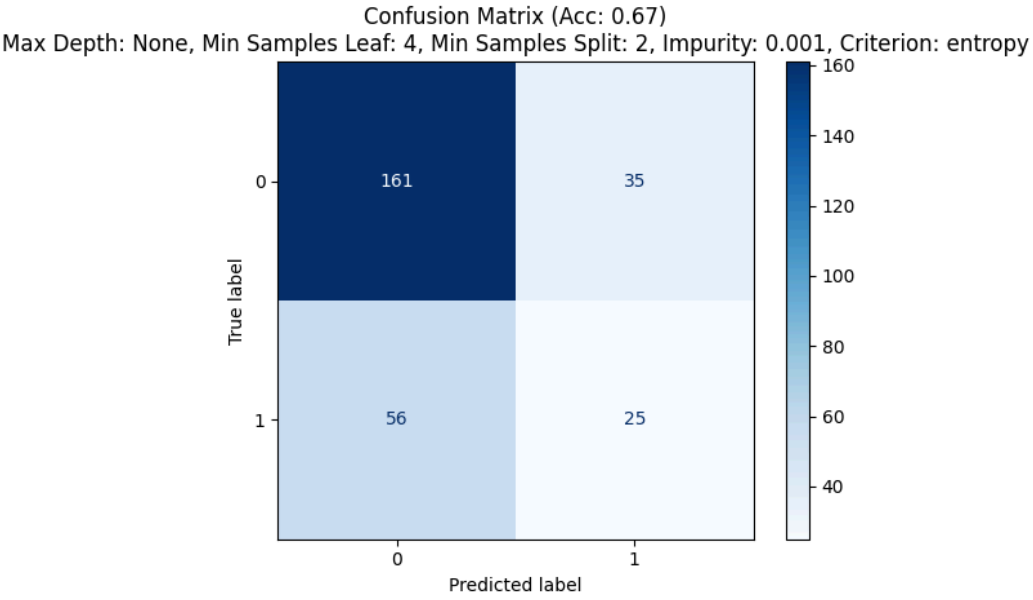
Third configuration: min_impurity_decrease=0.01, criterion='entropy', all the other params are set to default values
Confusion Matrix and Classification Report:



	precision	recall	f1-score	support
0	0.76	0.93	0.83	196
1	0.62	0.28	0.39	81
accuracy			0.74	277
macro avg	0.69	0.61	0.61	277
weighted avg	0.72	0.74	0.70	277

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

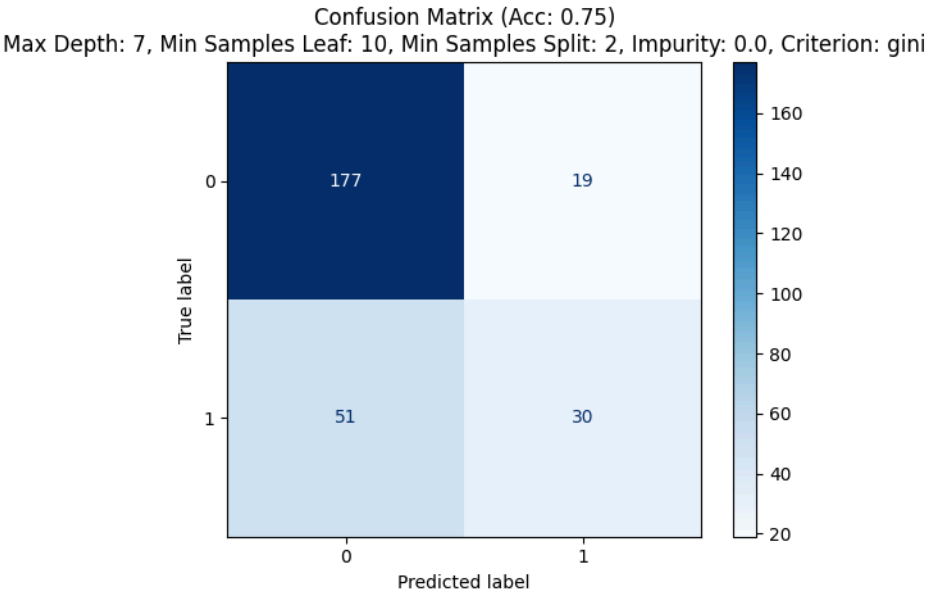
Confusion Matrix and Classification Report:



	precision	recall	f1-score	support
0	0.74	0.82	0.78	196
1	0.42	0.31	0.35	81
accuracy			0.67	277
macro avg	0.58	0.57	0.57	277
weighted avg	0.65	0.67	0.66	277

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

Confusion Matrix and Classification Report:



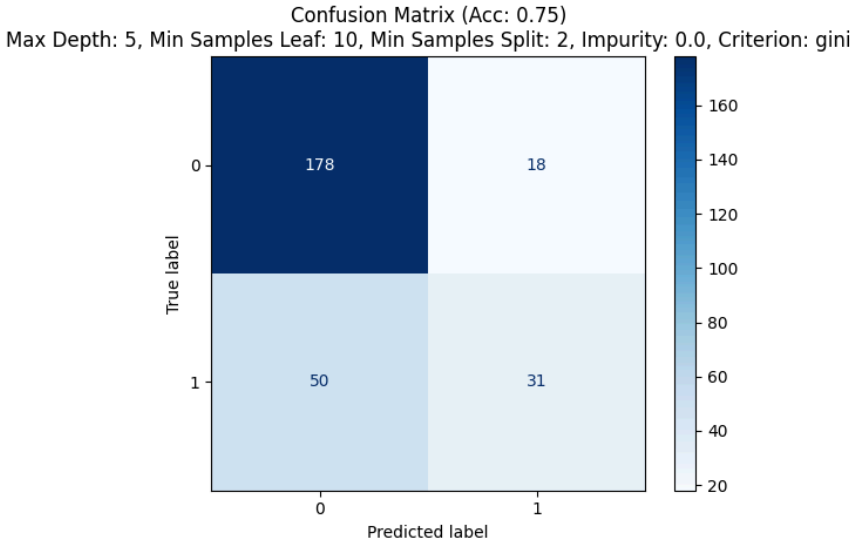
	precision	recall	f1-score	support
0	0.78	0.90	0.83	196
1	0.61	0.37	0.46	81
accuracy			0.75	277
macro avg	0.69	0.64	0.65	277
weighted avg	0.73	0.75	0.73	277

At the end, I performed a Grid Search on the 10 fold Cross Validation to find the best suitable parameters:

```python
#Grid Search
from sklearn.model_selection import GridSearchCV

param_grid = {
    'max_depth': [3, 5, 7, 9, None],
    'min_samples_leaf': [1, 2, 4, 10, 15, 20],
    'min_samples_split': [2, 5, 10, 15, 20],
    'min_impurity_decrease': [0.0, 0.001, 0.005, 0.01],
    'criterion': ['gini', 'entropy']
}

clf = DecisionTreeClassifier(random_state=42)
grid_search = GridSearchCV(clf, param_grid, cv=10)
grid_search.fit(X, y)

print(grid_search.best_params_)
```

The best parameters are: **{'criterion': 'gini', 'max_depth': 5, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 10, 'min_samples_split': 2}**

Sixth Configuration using Grid Search suggested parameters
Confusion Matrix and Classification Report:



	precision	recall	f1-score	support
0	0.78	0.91	0.84	196
1	0.63	0.38	0.48	81
accuracy			0.75	277
macro avg	0.71	0.65	0.66	277
weighted avg	0.74	0.75	0.73	277

This is the best performance reachable with this dataset, being it very unbalanced.

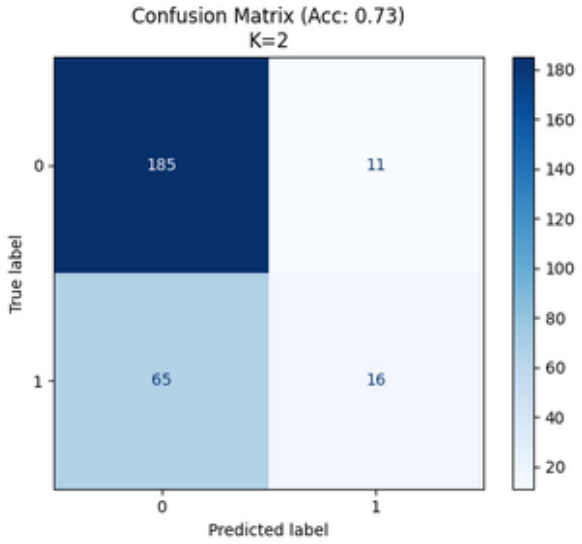
Question N.4 K-Nearest Neighbors and Naive Bayes Classifier
In order to apply the Stratified 10 Fold Cross Validation and assess the influence of the value of K on the average accuracy I've coded this function:

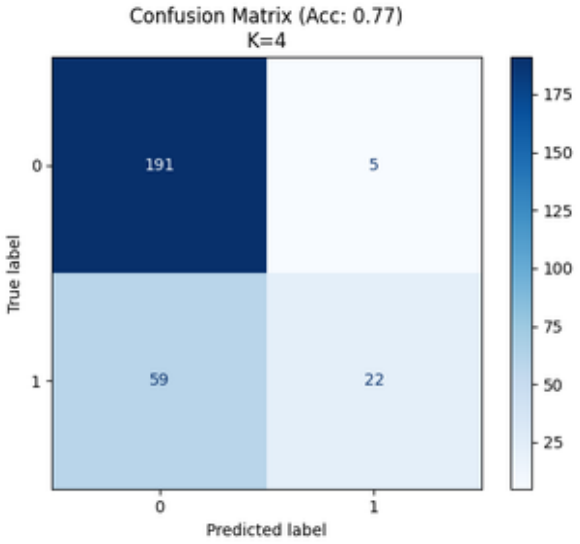
```python
def evaluate_knn_with_cv(X, y, n_neighbors=5):
    """
    Train a K-NN classifier with a specific K and perform 10-fold stratified cross-validation.
    """
    # Initialize the K-NN classifier
    knn = KNeighborsClassifier(n_neighbors=n_neighbors)

    # Perform 10-fold Stratified CV
    skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
    y_pred = cross_val_predict(knn, X, y, cv=skf)
    accuracy = accuracy_score(y, y_pred)
    prec = precision_score(y, y_pred)

    # Generate confusion matrix
    cm = confusion_matrix(y, y_pred)
    ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y)).plot(cmap='Blues')
    plt.title(f"Confusion Matrix (Acc: {accuracy:.2f})\nK={n_neighbors}")
    plt.show()

    return accuracy
```

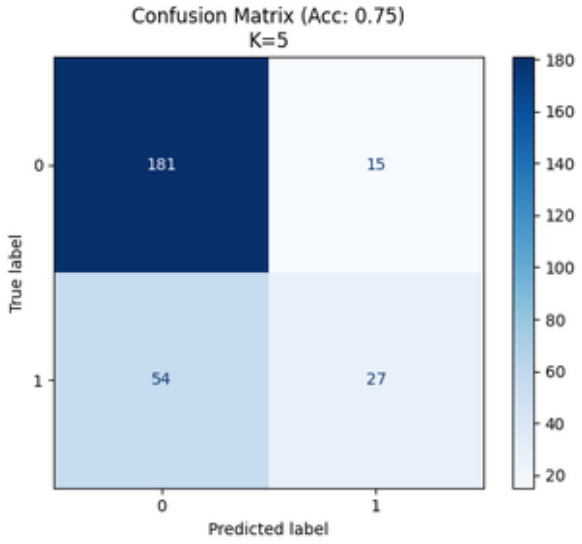
I've then created a linear space of K going from 1 (only one neighbor) to 20 (20 neighbors) with a step of 1 and run a for cycle to evaluate all the accuracies and show the confusion matrices:

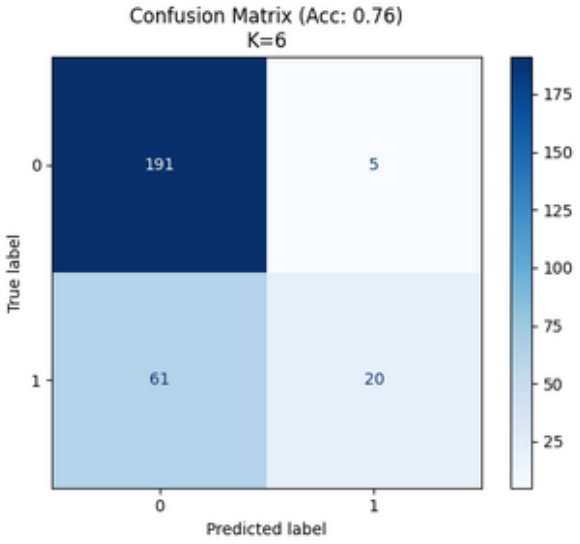
```python
k_values = np.linspace(1, 20, 20, dtype=int)
print(k_values)
accuracies = {}

for k in k_values:
    print(f"Evaluating K={k}")
    accuracies[k] = evaluate_knn_with_cv(X, y, n_neighbors=k)
```

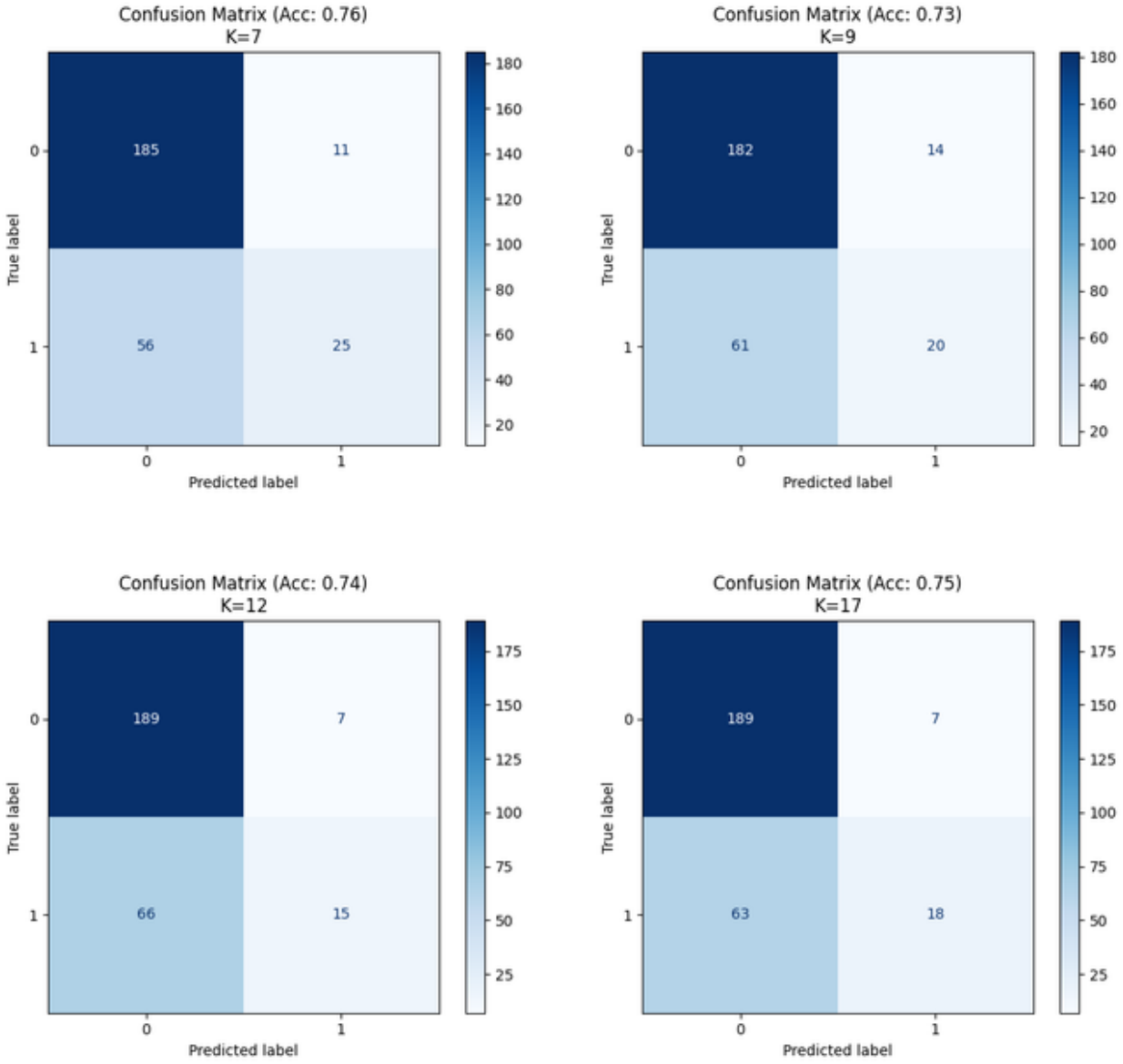
I'd now report some of the most significative confusion matrices among the ones observed:


Confusion Matrix (Acc: 0.73)
K=2
True label
Predicted label
185
11
65
16
0
1
180
160
140
120
100
80
60
40
20

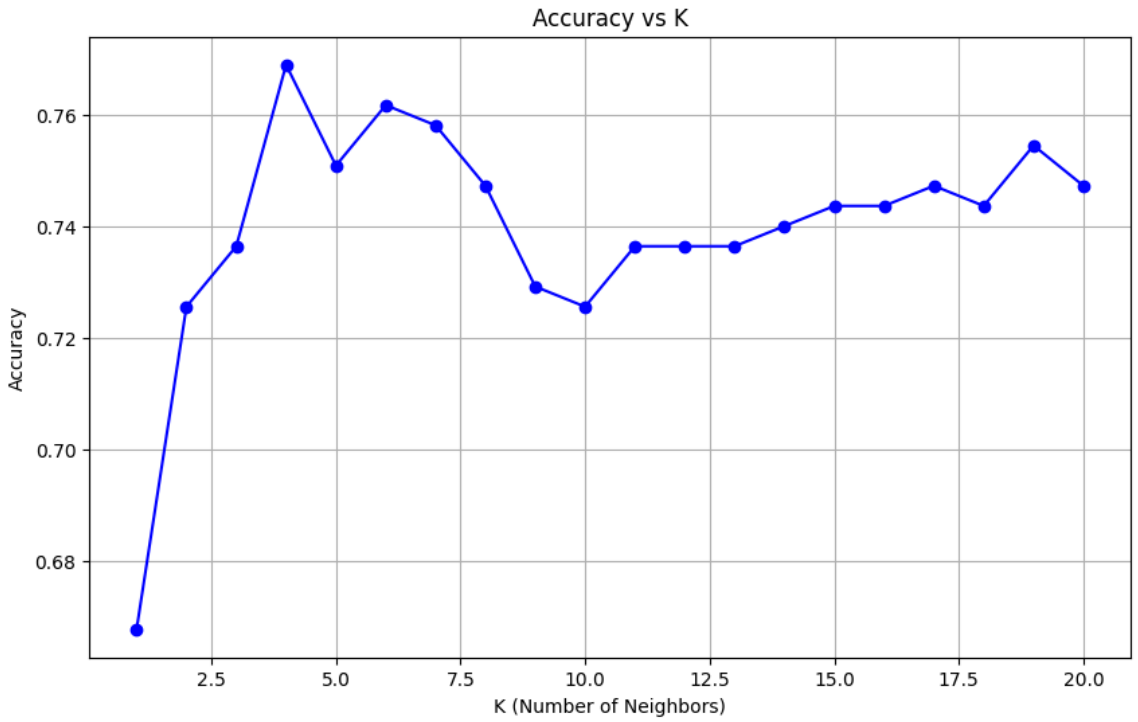

Confusion Matrix (Acc: 0.77)
K=4
True label
Predicted label
191
5
59
22
0
1
175
150
125
100
75
50
25


Confusion Matrix (Acc: 0.75)
K=5
True label
Predicted label
181
15
54
27
0
1
180
160
140
120
100
80
60
40
20


Confusion Matrix (Acc: 0.76)
K=6
True label
Predicted label
191
5
61
20
0
1
175
150
125
100
75
50
25



This chart show, for every K used to train the Classifier, the average accuracy observed with the 10 fold CV:

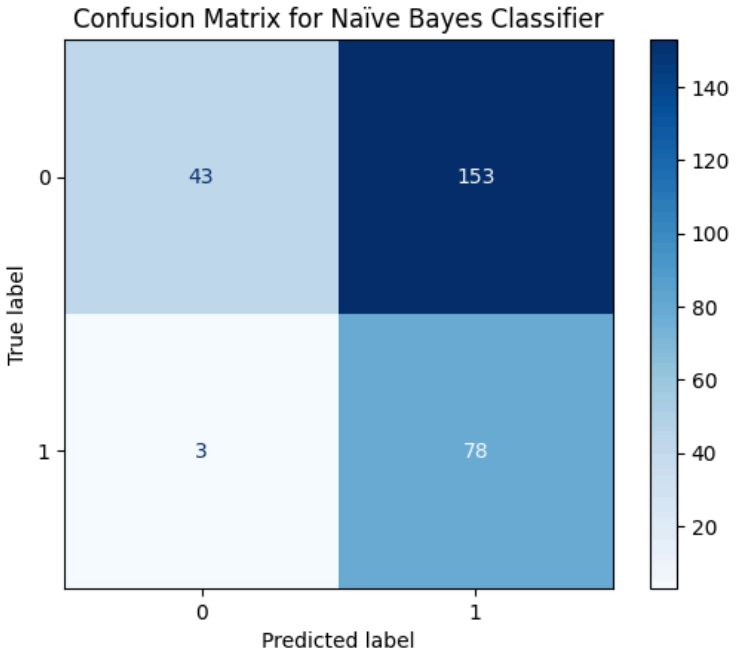


If K is very low (K < 4) the accuracy tends to be low. The **highest accuracy** is obtained setting **K=4** (Acc = 77%). Approximately for K > 7 we would expect to have a decreasing accuracy. This happens for K in [8; 10], but then the accuracy stabilizes and (curiously) grows again a little. I think this is a quite unexpected behavior, since the dataset has roughly 277 rows after clearing. It's evident that due to the strong imbalance of the two classes, intermediate values of K (like K = 9 or K = 10) lead to higher classification errors. On the other hand, higher values of K (like K > 10) could in this case balance bias and variance in a better way.

Comparison with Naïve Bayes Classifier

I've then applied the same approach with the Naïve Bayes Classifier:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.metrics import confusion_matrix, accuracy_score

# Modello Naïve Bayes
naive_bayes_model = GaussianNB()
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# 10-Fold Cross-Validation con accuratezza
cv_scores = cross_val_score(naive_bayes_model, X, y, cv=skf, scoring='accuracy')

# Predizioni per la matrice di confusione
y_pred_cv = cross_val_predict(naive_bayes_model, X, y, cv=skf)
# Generate confusion matrix
cm = confusion_matrix(y, y_pred_cv)
ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y)).plot(cmap='Blues')
plt.title(f"Confusion Matrix for Naïve Bayes Classifier")
plt.show()
# Media delle accuratezze
average_accuracy_nb = cv_scores.mean()

print(f"Naïve Bayes Average Accuracy: {average_accuracy_nb:.2f}")
```



The accuracy for the model is just 44%, very low. However, I think that's worth a look...
At first, let's show the Classification Report for the model:

	precision	recall	f1-score	support
0	0.93	0.22	0.36	196
1	0.34	0.96	0.50	81
accuracy			0.44	277
macro avg	0.64	0.59	0.43	277
weighted avg	0.76	0.44	0.40	277

If you compare this with the resport generated for the other models (Decision Tree, K-NN) used, surely the recall values for the Classes are very very different from the previous ones. More precisely, if before we had a significantly higher **Recall** related to Class 0 (the dominant Class in the dataset with 196 rows against 81 belonging to Class 1), now it is exactly **the opposite**!
Having in mind the mathematical definition for Recall:

$$\text{Recall (r)} = \frac{\text{Number of objects correctly assigned to C}}{\text{Number of objects belonging to C}}$$

It is surprising that this model, on average during 10-fold cross-validation, correctly assigns almost all rows actally belonging to the Minority Class but fails to do so for the majority Class. Why is that?
I personally think that Naïve Bayes Classifier is hugely **influenced by choosing One Hot Encoding** as the Encoding Algorithm for categorical features:

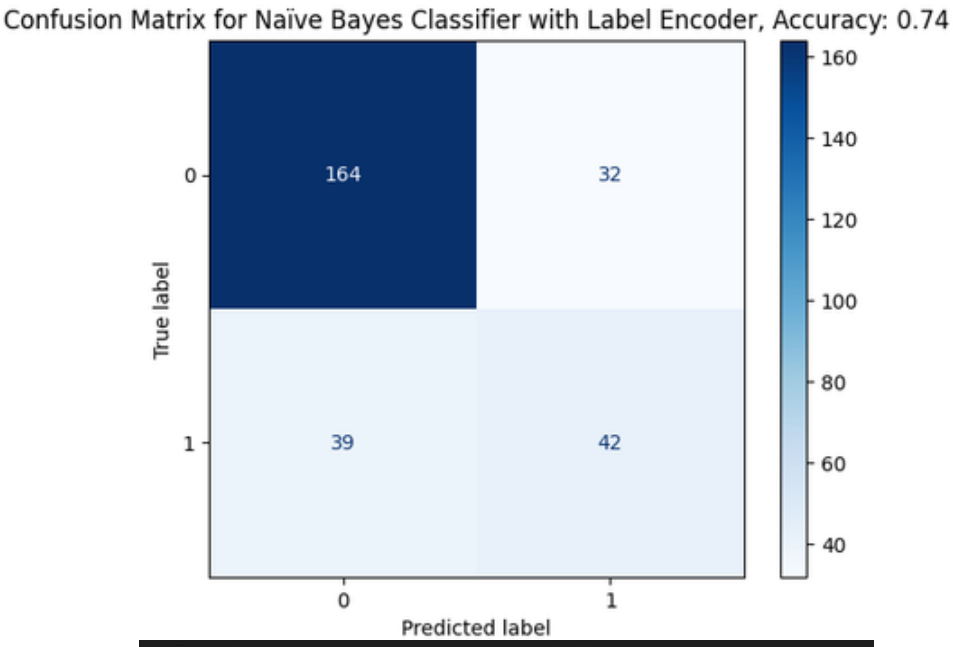
- Naïve Bayes works based on the assumption of conditional independence between the features, by using One Hot Encoding I create a dataset characterized by a sparse representation of either 1 or 0. This can **amplify the relative weight of certain features**

belonging to the minority Class, because the model calculates probabilities based on the occurrence of these features.

- Also, if for some reason the features belonging to the minority Class are quite **distinctive**, the Classifier can assign **higher probabilities** to them and be more biased towards that Class.
- At last but not least, if the minority Class has many features **overlappping** with ones belonging to the majority Class, the model can misinterpret the latter, returning as a result a lower recall.

I think all of this problems can be overcome by using the **Label Encoder** instead of the One Hot Encoder when applying a Naïve Bayes Classifier:

```
#Naive Bayes with Label Encoding
naive_bayes_model = GaussianNB()
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
# 10-Fold Cross-Validation con accuratezza
cv_scores = cross_val_score(naive_bayes_model, X, y, cv=skf, scoring='accuracy')

# Predizioni per la matrice di confusione
y_pred_cv = cross_val_predict(naive_bayes_model, X, y, cv=skf)

# Media delle accuratezze
average_accuracy_nb = cv_scores.mean()

cm = confusion_matrix(y, y_pred_cv)
ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y)).plot(cmap='Blues')
plt.title(f"Confusion Matrix for Naïve Bayes Classifier with Label Encoder, Accuracy: {average_accuracy_nb:.2f}")
plt.show()

print(f"Naïve Bayes Average Accuracy: {average_accuracy_nb:.2f}")
print("classification_report: ", classification_report(y, y_pred_cv))
```



Naïve Bayes Average Accuracy: 0.74

	precision	recall	f1-score	support
0	0.81	0.84	0.82	196
1	0.57	0.52	0.54	81
accuracy			0.74	277
macro avg	0.69	0.68	0.68	277
weighted avg	0.74	0.74	0.74	277

As a result, now the average accuracy is **74%**, comparable with the performances achieved by the K-NN Classifier.
More precisely, the True Positive are slighyl lower with respect to the average performance of K-NN Classifier, whereas the True Negative are actually higher than the False Negatives! So, this proves that the Naïve Bayes, even if Label Encoding is applied, **weights the minority Class relatively better than all the other models**, especially K-NN.