PW5-Supervised Machine Learning : Decision Tree Classification Algorithm

1 Summary

Decision trees are versatile Machine learning algorithms able to perform both classification and regression tasks. There are able to to fit complex data. There are the fundamental components of random forests which are among the most powerful Machien Learning algorithms available today. In this PW we start with discussing how to train, visualize and make prediction with simple decision trees.

2 Exercises

Exercise 1 : classification of Iris Images

In this exercise we start with a simple dataset widely used in the literature and well known by ChatGPT. Nevertheless, you will have time to do it by your self. The famous dataset of Iris contains the sepal and petal length and width of 150 iris flowers of three different species: Iris Setosa, Iris versicolor, and Iris virginica (Iris Dataset). As you can learn from this dataset description, we have three classes of Iris and we propose a decision tree model to train and predict new Iris.

1. load from sklean.datasets the Iris dataset after importing load_iris data object and use a variable-name $iris$ to store the returned value: iris = load_iris();
2. extract from iris.data the petal length and width then affect them to an input data as a new variable X. Note that the different columns or features of iris are the following:
   - Length of the sepal (in cm)
   - Width of the sepal (in cm)
   - Length of the petal (in cm)
   - Width of the petal (in cm)
3. print the target variable's names, values and count the number of classes.

4. import DecisionTreeClassifier from sklearn.tree and create a new object **treeClassifer** with a maximum depth value equal to 2.

max_depth decision tree

- The max_depth parameter is a key hyperparameter in decision trees. It controls the maximum depth of the tree corresponding to the maximum number of levels the tree is allowed to have, counting from the root node to the leaf nodes. Then the decision tree construction algorithm stops splitting nodes beyond this limit. Thus, each path from the root to a leaf cannot exceed this depth.

- Limiting tree depth is a common technique for avoiding over-fitting in decision tree models. An unconstrained decision tree can become very complex and adapt very strongly to the training data, which can prevent it from generalizing well to new data.

- Bias-Variance Tradeoff: Bias-variance trade-off: by controlling tree growth, max_depth can contribute to finding a balance between bias and variance. A tree that is too shallow risks being under-adjusted (high bias, low variance), while one that is too deep risks being over-adjusted (low bias, high variance). Choosing the optimum depth strikes a balance between the two.

- it is often advisable to adjust max_depth using cross-validation or another hyperparameter optimizer to ensure optimal model performance on a validation dataset.

5. fit your decision tree on your data

6. visualize your decision tree using "export_graphviz()".This method provides a graph definition file in a ".dot" format. Name this file **"Iris_DTree.dot"**. Below, you find the core definition of the function ***"export_graphviz()"***:

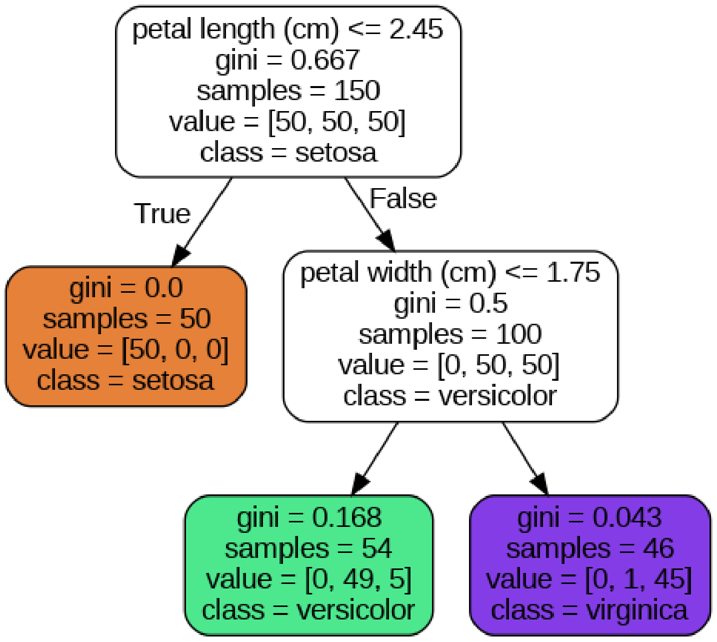
```python
 #imports the export_graphviz function from the Scikit-learn library, which is used to generate a GraphViz representation of the decision tree.
from sklearn.tree import export_graphviz
#imports the Image class from IPython's display module, which can be used to display images in a Jupyter Notebook or IPython environment.
from IPython.display import Image
#generates a GraphViz representation of the decision tree called "dot".
dot=export_graphviz(tree_1,
               out_file="Iris_DTree.dot", #specifies the output file name for the GraphViz representation. In this case, the tree will be saved as a file named "Iris_DTree.dot".
               feature_names=iris.feature_names[2:], #specifies the feature names for the tree visualization. It uses the feature names starting from the third feature (index 2) of the Iris dataset.
               class_names=Y_name, #specifies the class names for the tree visualization.
               rounded=True, #is an additional option to round the nodes and fill them with colors based on class.
               filled=True); #option


```

This code generates a GraphViz representation of the decision tree and saves it as a .dot file named "Iris_DTree.dot" in the current directory. To visualize the tree, you might need to convert this **.dot** file to an image format using GraphViz software.

7. call the command online as following:

```
#!It allows executing shell commands directly from the notebook.
!dot -Tpng Iris_DTree.dot -o Iris_DTree.png
#-T: type; -o: output
```

this command is executed in a Jupyter Notebook cell with the character "!" as a prefix. It takes the ".dot" file (Iris_DTree.dot) as input, converts it to a PNG image using the dot tool, and saves the resulting image as Iris_DTree.png. This enables visualization of the decision tree in a graphical format. Normaly you have the image below displayed on your Notebook.



Exercise 2: Decision Tree Interpretation

From the graph obtained on the Isis dataset, we obtained a tree with a level of deep equal to 2 when we include the root as a starting level (level of root equal to 0 and the last level of leafs is equal to 2). If we take any node in the tree, it contains the following information:

- petal width (cm) ≤ 0.8: corresponds to the ***petal*** feature selected based on Gini criteria.
- Gini criteria: it measure the impurity of a node according a selected feature. Indeed the Gini criteria is a metric in 0-1 range and calculated based on the number of samples having the same label according to the selected label.
- Samples: is the number of samples verifying the Gini criteria.

- Value: is vector of shape equal to $(1, classes)$ Class: is the label affected to the node and corresponds to the major label of the samples belonging to this class

1. calculate the Gini value for both features and verify that the petal length feature is well selected as a root node. We recall the Gini impurity formula here: $Gini_i = 1 - \sum_{k=1}^{n} p_{i,k}^2$ where $p_{i,k}$ is the ratio of class k instances among the training instances in the i^{th} node. The gini value of the root is: $1 - 3 \times (50/150)^2 = 0.667$

2. A decision tree can also estimate the probability that an instance belongs to a particular class k. Suppose you hve found a flower whose petals are 5 cm long and 1.5 cm on width. Use the $tree$ to estimate the probability to belong to each class. Note: you can use the function predict_proba().

3. interpreat the probabilities estimated before using the tree graph to extract the path. Rouhphly speaking, you have to start asking the root node whether the petal length of the flower is smaller or greater than 0.8. Depending on the aswer you will move down to the root's left child or to the right one. You have to do the same process until you reach the final leaf. The final sheet will be the one with the highest estimated probability amongst the 3.

4. Verify the last result using the function predict.

exercise 4: k-fold cross validation

Now we are interesting to evaluate the performance of our tree by splitting data into training and test. We introduce in this exercise the k-fold cross validation technique where a dataset is divided into k subsets or folds. The model is trained and evaluated k times, using a different fold as the validation set each time. Performance metrics from each fold are averaged to estimate the model's generalization performance. Here is a simple implementation of the k-fold cross-validation The algorithm is the following

```
Function kFoldCrossValidation:
 Input:
     - Dataset X
     - Machine Learning Model M
     - Number of folds k

 Output:
     - Average performance metric

 Divide X into k equal-sized subsets (folds)

 Initialize an empty list to store performance metrics

 for each fold f in k:
     Split X into training set (X_train) and validation set (X_val)
         X_train = X\ fold f X minus fold k
         X_val = fold f

     Train M on X_train
     Evaluate M on X_val to compute performance metric (e.g., accuracy, error)

```

```
        Store the performance metric in the list

    Calculate the average performance metric over the k iterations

    Return the average performance metric
```

```
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
def kfoldCrossValidation(X,y,M):
  # Set up k-fold cross-validation
  kfold = KFold(n_splits=5, shuffle=True, random_state=42)
  scores = []

# Perform k-fold cross-validation
  for train_index, test_index in kfold.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Fit the classifier on the training data
    M.fit(X_train, y_train)

    # Predict on the test data
    y_pred = M.predict(X_test)

    # Calculate accuracy and store in scores list
    accuracy = accuracy_score(y_test, y_pred)
    scores.append(accuracy)
  return scores
```

1. Modify the function kfoldCrossValidation after including the k and the machine learning model M, as additional input parameters
2. implement the average accuracy from the scores array returned by the function kfoldCrossValidation
3. apply the average accuracy on your trained tree using different values of k
4. what is your intuition about the impact of the k on the model performance?
5. built an other decision tree using all the features without the max_depth constrain, and compare the average score with the precedent.

Exercise 5: Gini Or Entropy

By default, Gini impurity measure is used in scikitlearn. The Entropy impurity is a concept originated in thermodynamics as a measure of molecular disorder. It approaches 0 when molecules are well ordered. It is also a popular concept in information theory used by Shannon to leasure the average information content of a message. In Machine learning Entropy is used as an impurity measure set to 0 when it contains instances of only one class. It quantifies the average class uncertainty within a node. A high entropy means that classes are highly mixed, while a low entropy indicates high purity.

The Entropy formula is usually called $\mathcal{H}$: $-\sum_{k=1_{p_{i,k}\neq 0}}^{n} p_{i,k} log_2(p_{i,k})$. Gini tends to isolate the most frequent class on its own branch of the tree while Entropy tends to produce more balanced trees.

1. built a new tree using the hole features, the entropy as a metric, and random_state=42 (call this new decision tree as *dt_entropy*)
2. split the data set onto train and test considering 30% of samples for testing
3. train your tree by calling the function fit() on *dt_entropy*
4. evaluate the performance (the accuracy) of *dt_entropy* on the test dataset (Use accuracy = dt_entropy.score(X_test, y_test)) and display the accuracy (Use print(f'Accuracy using entropy: accuracy:.3f"))
5. apply kfoldCrossValidation function on *dt_entropy* and compare both results.
6. compare the average score with entropy VS. with Gini (exercise 4, question 5)

Annex

To display the decision tree you can use the export graph environment based on the format .dot or you can use matplotlib.pyplot to plot the tree as an image. Here are the two methods:

```
    from sklearn.tree import export_graphviz
from IPython.display import Image
dot=export_graphviz(dt_entropy,
                out_file="iris_Ent_tree.dot",
                feature_names=iris.feature_names,
                class_names=iris.target_names,
                rounded=True,
                filled=True);

!dot -Tpng iris_Ent_tree.dot -o iris_Ent_tree.jpg
from PIL import Image
# creating a object
image = Image.open(r"iris_Ent_tree.jpg")
#image.show() can't work correctly so we use display instead !
display(image);
```

The second method is:

```
# plot the descision tree
plt.figure(figsize=(12, 8))
plot_tree(dt_entropy, filled=True, feature_names=iris.feature_names, class_names=iris.
    target_names)
plt.show()

#  valuer  les performances du mod le sur les donn es de test
accuracy = dt_entropy.score(X_test, y_test)
print(f"Accuracy using entropy: {accuracy:.4f}")



```