CS 422 Due:23h00
Data Mining Homework 2 March 23rd, 2025

Practicum Problems

These problems will primarily reference the lecture materials and examples provided in class using Python. It is recommended that a Jupyter/IPython notebook be used for the programmatic components. Students are expected to refer to the prescribed textbook or credible online resources to answer the questions accurately.

NAME:Zizhuo Shi A#:A20563449

Problem 1

Load the Iris sample dataset from sklearn (using load_iris()) into Python with a Pandas DataFrame. Induce a set of binary decision trees with a minimum of 2 instances in the leaves (min_samples_leaf=2), no splits of subsets below 5 (min_samples_split=5), and a maximum tree depth ranging from 1 to 5 (max_depth=1 to 5). You can leave other parameters at their default values. Which depth values result in the highest Recall? Why? Which value resulted in the lowest Precision? Why? Which value results in the best F1 score? Also, explain the difference between the micro, macro, and weighted methods of score calculation

According to the requirements of the topic, I downloaded iris dataset in my python code and recorded the classification report at each depth. The results are as follows

Max Depth: 1
Classification Report:

	precision	recall	f1-score	support
0	1.000000	1.000000	1.000000	50.000000
1	0.500000	1.000000	0.666667	50.000000
2	0.000000	0.000000	0.000000	50.000000
accuracy	0.666667	0.666667	0.666667	0.666667
macro avg	0.500000	0.666667	0.555556	150.000000
weighted avg	0.500000	0.666667	0.555556	150.000000

==========

Max Depth: 2
Classification Report:

	precision	recall	f1-score	support
0	1.000000	1.00	1.000000	50.00
1	0.907407	0.98	0.942308	50.00
2	0.978261	0.90	0.937500	50.00
accuracy	0.960000	0.96	0.960000	0.96
macro avg	0.961889	0.96	0.959936	150.00
weighted avg	0.961889	0.96	0.959936	150.00

==========

Max Depth: 3
Classification Report:

	precision	recall	f1-score	support
0	1.000000	1.000000	1.000000	50.000000
1	0.979167	0.940000	0.959184	50.000000
2	0.942308	0.980000	0.960784	50.000000
accuracy	0.973333	0.973333	0.973333	0.973333
macro avg	0.973825	0.973333	0.973323	150.000000
weighted avg	0.973825	0.973333	0.973323	150.000000

==========

Max Depth: 4
Classification Report:

	precision	recall	f1-score	support
0	1.000000	1.00	1.000000	50.00
1	0.960784	0.98	0.970297	50.00
2	0.979592	0.96	0.969697	50.00
accuracy	0.980000	0.98	0.980000	0.98
macro avg	0.980125	0.98	0.979998	150.00
weighted avg	0.980125	0.98	0.979998	150.00

==========

Max Depth: 5
Classification Report:

	precision	recall	f1-score	support
0	1.000000	1.00	1.000000	50.00
1	0.960784	0.98	0.970297	50.00
2	0.979592	0.96	0.969697	50.00
accuracy	0.980000	0.98	0.980000	0.98
macro avg	0.980125	0.98	0.979998	150.00
weighted avg	0.980125	0.98	0.979998	150.00

E.N.D

Based on these data and the macro method, I found that the depth of the maximum regression is 4.Based on the same approach, I found that the lowest precision in these classification report has a depth of 1, and the best F1 score has a depth value of 4, as shown below.

```
best_depth = max(recall_results, key=lambda k: recall_results[k]['macro avg']['recall'])
best_recall = recall_results[best_depth]['macro avg']['recall']
print(f"Best depth: {best_depth}, with Recall: {best_recall:.2f}")
```

```
Best depth: 4, with Recall: 0.98
```

```
best_depth = min(recall_results, key=lambda k: recall_results[k]['macro avg']['precision'])
best_precision = recall_results[best_depth]['macro avg']['precision']
print(f"Lowest Precision depth: {best_depth}, with Precision: {best_precision:.2f}")
```

```
Lowest Precision depth: 1, with Precision: 0.50
```

```
best_depth = max(recall_results, key=lambda k: recall_results[k]['macro avg']['f1-score'])
best_f1score = recall_results[best_depth]['macro avg']['f1-score']
print(f"Best F1 score depth: {best_depth}, with F1 score: {best_f1score:.2f}")
```

```
Best F1 score depth: 4, with F1 score: 0.98
```

The three methods will be distinguished from the two aspects of definition and calculation method.

Firstly,the micro method calculates the metrics globally by counting the total true positives, false positives, and false negatives across all classes.It sums up the true positives, false positives, and false negatives of all classes and then computes the metric using these aggregated values.

Second,the macro method calculates the metrics for each class independently and then takes the average of the metrics across all classes.It computes the metric for each class separately and then takes the unweighted mean of these metrics.

What's more, the weighted method calculates the metrics for each class independently and then averages the metrics weighted by the number of instances in each class.It computes the metric for each class separately and then averages these metrics, weighted by the support (number of instances) for each class.

All in all, Micro is a global calculation and all instances are treated equally.Macro is a per-class calculation and all classes are treated equally.Weighted is also a per-class calculation, but it will be weighted according to the number of category samples.

Problem 2

Load the Breast Cancer Wisconsin (Diagnostic) sample dataset from the UCI Machine Learning Repository (the discrete version at: breast-cancer-wisconsin.data) into Python using a Pandas DataFrame. Induce a binary Decision Tree with a minimum of 2 instances in the leaves, no splits of subsets below 5, and a maximum tree depth of 2 (using the default Gini criterion). Calculate the Entropy, Gini, and Misclassification Error of the first

E.N.D

split. What is the Information Gain? Which feature is selected for the first split, and what value determines the decision boundary?

First of all, according to the meaning of the topic, I define the calculation entory,gini impurity and misclassification error function, and get three values before split.

```
# Calculate Gini index for the entire dataset
def gini_index(y): # 1-∑(pi)^2
    p = y.value_counts(normalize=True)
    return 1 - np.sum(np.square(p))

# Calculate metrics for the entire dataset
gini_before_split = gini_index(y_train)
print(f"Gini before split: {gini_before_split}")
```

```
Gini before split: 0.46693877551020413
```

```
# Calculate Entropy for the entire dataset
def entropy(y): #-∑pi*log2*pi*
    p = y.value_counts(normalize=True)
    return -np.sum([p_i * np.log2(p_i) for p_i in p])

entropy_before_split = entropy(y_train)
print(f"Entropy before split: {entropy_before_split}")
```

```
Entropy before split: 0.9164534336173732
```

```
# Calculate Misclassification Error for the entire dataset
def misclassification_error(y):# 1-max(pi)
    p = y.value_counts(normalize=True)
    return 1 - np.max(p)

misclassification_error_before_split = misclassification_error(y_train)
print(f"Misclassification Error before split: {misclassification_error_before_split}")
```

```
Misclassification Error before split: 0.33150183150183155
```

By building the tree and the calculation results after the first split, we get the data after the first spilt.

```
Gini after split: 0.0
Entropy after split: -0.0
Misclassification Error after split: 0.0
```

The Information Gain is 0.44321673442552567 and class is selected for the first split, and the decision boundary value is 0.5.

```
# Calculate Information Gain
information_gain = gini_before_split - (len(left_split) / len(X_train)) * gini_left - (len(right_split) / len(X_train)) * gini_right

print(f"Information Gain for the first split: {information_gain}")
```

```
Information Gain for the first split: 0.44321673442552567
```

```
# Map feature index to feature name
first_split_feature_name = X.columns[first_split_feature]

print(f"The feature selected for the first split is: {first_split_feature_name}")
print(f"The decision boundary value for the first split is: {first_split_threshold}")
```

```
The feature selected for the first split is: Class
The decision boundary value for the first split is: 0.5
```

E.N.D

CS 422 Due:23h00
Data Mining Homework 2 March 23rd, 2025

Problem 3

Load the Breast Cancer Wisconsin (Diagnostic) sample dataset from the UCI Machine Learning Repository (the continuous version at: wdbc.data) into Python using a Pandas DataFrame. Induce the same binary Decision Tree as above (now using the continuous data), but perform PCA dimensionality reduction beforehand. Using only the first principal component of the data for model fitting, what are the F1 score, Precision, and Recall of the PCA-based single factor model compared to the original (continuous) data? Repeat the process using the first and second principal components. Using the Confusion Matrix, what are the values for False Positives (FP) and True Positives (TP), as well as the False Positive Rate (FPR) and True Positive Rate (TPR)? Is using continuous data beneficial for the model in this case? How?"

After deciding that the tree uses the first principle component, we can get the PCA-based model's F1 score, Precision, and Recall .Using the full set of continuous features, we can get the Original model's F1 score, Precision, and Recall.

```
# Train the decision tree using the first principal component
clf_pca = DecisionTreeClassifier(
    min_samples_leaf=2,
    min_samples_split=5,
    max_depth=2,
    criterion='gini'
)

clf_pca.fit(X_train_pca, y_train)

# Predict on the test set
y_pred_pca = clf_pca.predict(X_test_pca)

# Calculate F1 score, Precision, and Recall
f1_pca = f1_score(y_test, y_pred_pca)
print(f"F1 Score (PCA-based model): {f1_pca}")

precision_pca = precision_score(y_test, y_pred_pca)
print(f"Precision (PCA-based model): {precision_pca}")

recall_pca = recall_score(y_test, y_pred_pca)
print(f"Recall (PCA-based model): {recall_pca}")
```

```
F1 Score (PCA-based model): 0.9523809523809523
Precision (PCA-based model): 0.975609756097561
Recall (PCA-based model): 0.9302325581395349
```

E.N.D

```
# Train the decision tree using the original data
clf_original = DecisionTreeClassifier(
    min_samples_leaf=2,
    min_samples_split=5,
    max_depth=2,
    criterion='gini'
)

clf_original.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred_original = clf_original.predict(X_test_scaled)

# Calculate F1 score, Precision, and Recall
f1_original = f1_score(y_test, y_pred_original)
print(f"F1 Score (Original model): {f1_original}")

precision_original = precision_score(y_test, y_pred_original)
print(f"Precision (Original model): {precision_original}")

recall_original = recall_score(y_test, y_pred_original)
print(f"Recall (Original model): {recall_original}")
```

```
F1 Score (Original model): 0.9024390243902439
Precision (Original model): 0.9487179487179487
Recall (Original model): 0.8604651162790697
```

Using the Confusion Matrix, the values for False Positives (FP) is 1, True Positives (TP) is 38, the False Positive Rate (FPR) is 0.014084507042253521, and True Positive Rate (TPR) is 0.8837209302325582

```
# Extract FP, TP, FN, TN from the Confusion Matrix
FP = cm_pca[0, 1]
TP = cm_pca[1, 1]
FN = cm_pca[1, 0]
TN = cm_pca[0, 0]

# Calculate FPR and TPR
FPR = FP / (FP + TN)
TPR = TP / (TP + FN)

print(f"False Positives (FP): {FP}")
print(f"True Positives (TP): {TP}")
print(f"False Positive Rate (FPR): {FPR}")
print(f"True Positive Rate (TPR): {TPR}")
```

```
False Positives (FP): 1
True Positives (TP): 38
False Positive Rate (FPR): 0.014084507042253521
True Positive Rate (TPR): 0.8837209302325582
```

E.N.D

Repeat the process ,we can get the value.

```
# Train the decision tree using the original data
clf_original = DecisionTreeClassifier(min_samples_leaf=2, min_samples_split=5, max_depth=2, criterion='gini')
clf_original.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred_original = clf_original.predict(X_test_scaled)

# Calculate the Confusion Matrix for the original model
cm_original = confusion_matrix(y_test, y_pred_original)

# Extract FP, TP, FN, TN from the Confusion Matrix
FP_original = cm_original[0, 1]
TP_original = cm_original[1, 1]
FN_original = cm_original[1, 0]
TN_original = cm_original[0, 0]

# Calculate FPR and TPR for the original model
FPR_original = FP_original / (FP_original + TN_original)
TPR_original = TP_original / (TP_original + FN_original)

print(f"False Positives (FP) - Original model: {FP_original}")
print(f"True Positives (TP) - Original model: {TP_original}")
print(f"False Positive Rate (FPR) - Original model: {FPR_original}")
print(f"True Positive Rate (TPR) - Original model: {TPR_original}")
```

```
False Positives (FP) - Original model: 2
True Positives (TP) - Original model: 37
False Positive Rate (FPR) - Original model: 0.028169014084507043
True Positive Rate (TPR) - Original model: 0.8604651162790697
```

Based on the above data, we can conclude that using continuous data is more advantageous. Because it has improved precision,higher recall.

E.N.D