Machine Learning Assignment I: Collaborative and Adaptive Classifier Systems

Q1: Collaborative Multi-Classifier System Design

Description:

Using the Mini-ImageNet dataset, design a collaborative multi-classifier system that combines at least two different classifiers (e.g., k-NN, SVM, Decision Tree) to leverage their complementary strengths and enhance classification accuracy. Your system should be able to dynamically select the best classifier based on the characteristics of each sample.

Dataset Format:

The Mini-ImageNet dataset contains images and labels divided into training and test sets. The dataset includes (you can find it from kaggle)

- Training set: 32x32 RGB images, 100 classes, 500 images per class.
- Test set: 32x32 RGB images, 100 classes, 100 images per class.
- Label Format: Provided as a .csv file with 'image_path' and 'label' columns.
- Example:

```
image_path,label
/path/to/image1.jpg,5
/path/to/image2.jpg,22
```

Note: Original dataset from kaggle includes 100 classes, 600 images per class at different scales. You need to preprocess the dataset into the above format, including dataset splitting, image size, and label format. (If you don't know how to do that, please ask ChatGPT or Claude first. If it doesn't work, you can send an email to TA for detailed guidance.)

Task Requirements:

1. Multi-Classifier Combination:

- Implement two or more classifiers (e.g., k-NN with configurable k-value, SVM with adjustable C-value and kernel selection, Decision Tree with parameters like max_depth). Test your classifiers on Mini-ImageNet and combine them using a chosen strategy (e.g., weighted voting or selective classification based on sample features).
- **Example combination strategies:**
  - **Weighted Voting**: Assign weights to each classifier (e.g., k-NN weight 0.6, SVM weight 0.4) and use weighted voting to select the final class.
  - **Selective Classification**: Design a system where samples with specific feature patterns are directed to SVM, while others are directed to k-NN.

2. Classifier Selection Logic:

- - Design a simple function, `classifier_selector(image)`, to choose a classifier based on sample characteristics (e.g., average pixel intensity). For example:
  ```python
  def classifier_selector(image):
      mean_intensity = np.mean(image)
      if mean_intensity > threshold:
          return 'SVM'
      else:
          return 'k-NN'
  ```

3. Performance Evaluation:

- Evaluate the system's accuracy, precision, recall, and F1-score. Compare these metrics between the individual classifiers and the collaborative system.
- Implement an evaluation function:

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
def evaluate_performance(y_true, y_pred):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='macro')
    recall = recall_score(y_true, y_pred, average='macro')
    f1 = f1_score(y_true, y_pred, average='macro')
    return accuracy, precision, recall, f1
```

4. Report Analysis:

- In your report, explain the classifier selection logic, combination strategy, and experimental results comparing single vs. collaborative classifiers. Discuss complementary strengths and potential improvements.

Q2: Adaptive Classifier Adjustment System

Description:

Design an adaptive classifier system using the Mini-ImageNet dataset that can automatically adjust classifier selection and parameters based on different data distribution conditions, such as class imbalance or varying sample sizes. The system should dynamically select and tune classifiers for optimal performance under diverse conditions.

Task Requirements:

1. Adaptive Selection Logic:

- Develop a function `adaptive_classifier(image, label_distribution)` that selects a classifier and adjusts its parameters based on the data conditions (e.g., class imbalance or ample sample size).
- Example:

```python
def adaptive_classifier(image, label_distribution):
    # Dynamically choose based on distribution skewness
    if label_distribution.skewness > threshold:
        return 'SVM', best_C_value
    else:
        return 'k-NN', best_k_value
```

2. Dynamic Parameter Adjustment:

- Implement parameter tuning strategies for SVM and k-NN based on data conditions, such as tuning the `C` value for SVM in imbalanced scenarios, or adjusting the `k` value for k-NN in dense data scenarios.
- Test the impact of these parameter changes on classification performance under varying sample distributions.

3. Performance Comparison:

- Evaluate the system's performance using accuracy, precision, recall, and F1-score, and compare adaptive vs. static systems.
- Analyze how adaptive tuning impacts these metrics in different scenarios, presenting the results in your report.

4. Report Analysis:

- In the report, explain the selection and tuning logic, experimental results, and the adaptive system's strengths and weaknesses across conditions.

Grading:

Submission Requirements:

- **Source Code**: Submit all source code files to Moddle, including the main program and any helper functions, with appropriate comments.
- **Report**: The report should include the following sections:
    - **Method Design**: Describe the logic behind adaptive classifier selection and the parameter adjustment strategies.
    - **Experimental Results**: Present the system's performance under different scenarios (include tables or graphs).
    - **Conclusion and Analysis**: Analyze the effects and impact of the system's adjustments across various data conditions.

Grading Criteria (Total: 100 Points):

1. **Adaptive Adjustment Design (30 points)**:

- Evaluation of the effectiveness of the classifier selection and adjustment logic according to different data conditions.

2. **Parameter Adjustment Strategy (20 points)**:
   - Assessment of the creativity and appropriateness of the parameter adjustment strategy.
3. **Performance Comparison (20 points)**:
   - Inclusion of performance comparisons across different scenarios, with reasonable explanations of the results.
4. **Report (20 points)**:
   - Completeness, logical coherence, and depth of system analysis within the report.
5. **Code Quality (10 points)**:
   - Clarity of code, adequacy of comments, and adherence to good programming practices.