Appendix-5: Random Forest

Random Forest is an ensemble learning algorithm that combines multiple decision trees to improve the classification accuracy and reduce overfitting. It works by creating random subsets of the data and features for each tree, then aggregating the predictions of all trees.

For our implementation, we used scikit-learn's RandomForestClassifier with 1000 trees and default hyperparameters.

In [1]:
```
# import library dependencies
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f
from sklearn.model_selection import train_test_split
import joblib
```

Import Data

In [2]:
```
ROOT_PATH='../'
```

In [3]:
```
# function to open pickle file
def unpickle(file):
    import pickle
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    return dict
```

In [4]:
```
# store each pickle files in individual batches
batch1 = unpickle(ROOT_PATH+"cifar-10-batches-py/data_batch_1")
batch2 = unpickle(ROOT_PATH+"cifar-10-batches-py/data_batch_2")
batch3 = unpickle(ROOT_PATH+"cifar-10-batches-py/data_batch_3")
batch4 = unpickle(ROOT_PATH+"cifar-10-batches-py/data_batch_4")
batch5 = unpickle(ROOT_PATH+"cifar-10-batches-py/data_batch_5")
test_batch = unpickle(ROOT_PATH+"cifar-10-batches-py/test_batch")
```

In [5]:
```
# function to create labels and images from data
def load_data0(btch):
    labels = btch[b'labels']
    imgs = btch[b'data'].reshape((-1, 32, 32, 3))

    res = []
    for ii in range(imgs.shape[0]):
        img = imgs[ii].copy()
        img = np.fliplr(np.rot90(np.transpose(img.flatten().reshape(3,32,32)
        res.append(img)
    imgs = np.stack(res)
    return labels, imgs
```

In [6]:
```
# function to load data into training and test set
def load_data():
    x_train_l = []
    y_train_l = []
    for ibatch in [batch1, batch2, batch3, batch4, batch5]:
        labels, imgs = load_data0(ibatch)
        x_train_l.append(imgs)
        y_train_l.extend(labels)
    x_train = np.vstack(x_train_l)
    y_train = np.vstack(y_train_l)

    x_test_l = []
    y_test_l = []
    labels, imgs = load_data0(test_batch)
    x_test_l.append(imgs)
    y_test_l.extend(labels)
    x_test = np.vstack(x_test_l)
    y_test = np.vstack(y_test_l)
    return (x_train, y_train), (x_test, y_test)
```

Preprocess Data

In [7]:
```
# create training and test set
(x_train, y_train), (x_test, y_test) = load_data()
```

In [8]:
```
print('x_train shape:', x_train.shape)
print('y_train shape:', y_train.shape)
print('x_test shape:', x_test.shape)
print('y_test shape:', y_test.shape)
```

```
x_train shape: (50000, 32, 32, 3)
y_train shape: (50000, 1)
x_test shape: (10000, 32, 32, 3)
y_test shape: (10000, 1)
```

In [9]:
```
print(x_train.shape[0], 'train samples (x)')
print(y_train.shape[0], 'train samples (y)')
```

```
50000 train samples (x)
50000 train samples (y)
```

In [10]:
```
print(x_test.shape[0], 'test samples (x)')
print(y_test.shape[0], 'test samples (y)')
```

```
10000 test samples (x)
10000 test samples (y)
```

In [11]:
```
# Flatten the images
X_train = x_train.reshape(x_train.shape[0], -1)
X_test = x_test.reshape(x_test.shape[0], -1)
```

In [12]:
```
# Normalize the data
X_train = X_train.astype('float32') / 255
X_test = X_test.astype('float32') / 255
```

In [13]:
```
# Reshape y_train and y_test to 1d arrays
y_train = y_train.ravel()
y_test = y_test.ravel()
```

In [14]:
```
# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_siz
```

Define model and train

In [15]:
```
# Define the model
rf = RandomForestClassifier(n_estimators=1000, n_jobs=-1, verbose=1)
```

In [16]:
```
# Train the model
rf.fit(X_train, y_train)
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    7.3s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   38.8s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:  1.5min
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed:  2.8min
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:  3.6min finished
```

Out[16]:

▾ RandomForestClassifier

```
RandomForestClassifier(n_estimators=1000, n_jobs=-1, verbose=1)
```

Save and load model

In [17]:
```
# Save the model to a file
joblib.dump(rf, 'random_forest.sav')
```

Out[17]:
```
['random_forest.sav']
```

In [18]:
```
# Load the saved model from a file
loaded_model = joblib.load('random_forest.sav')
```

Evaluate the model

In [19]:
```
# Evaluate the model
train_acc = accuracy_score(y_train, loaded_model.predict(X_train))
val_acc = accuracy_score(y_val, loaded_model.predict(X_val))
test_acc = accuracy_score(y_test, loaded_model.predict(X_test))

print(f"Train Accuracy: {train_acc:.4f}")
print(f"Val Accuracy: {val_acc:.4f}")
print(f"Test Accuracy: {test_acc:.4f}")
```

```
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent worke
rs.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.1s
[Parallel(n_jobs=8)]: Done 184 tasks      | elapsed:    0.8s
[Parallel(n_jobs=8)]: Done 434 tasks      | elapsed:    1.8s
[Parallel(n_jobs=8)]: Done 784 tasks      | elapsed:    3.3s
[Parallel(n_jobs=8)]: Done 1000 out of 1000 | elapsed:    4.3s finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent worke
rs.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.0s
[Parallel(n_jobs=8)]: Done 184 tasks      | elapsed:    0.2s
[Parallel(n_jobs=8)]: Done 434 tasks      | elapsed:    0.5s
[Parallel(n_jobs=8)]: Done 784 tasks      | elapsed:    0.8s
[Parallel(n_jobs=8)]: Done 1000 out of 1000 | elapsed:    1.0s finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent worke
rs.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.0s
[Parallel(n_jobs=8)]: Done 184 tasks      | elapsed:    0.2s
[Parallel(n_jobs=8)]: Done 434 tasks      | elapsed:    0.4s
[Parallel(n_jobs=8)]: Done 784 tasks      | elapsed:    0.8s
Train Accuracy: 1.0000
Val Accuracy: 0.4887
Test Accuracy: 0.4888
[Parallel(n_jobs=8)]: Done 1000 out of 1000 | elapsed:    1.0s finished
```

In [20]:

```python
# Calculate precision, recall, and F1 score on test set
test_pred = loaded_model.predict(X_test)
precision = precision_score(y_test, test_pred, average='weighted')
recall = recall_score(y_test, test_pred, average='weighted')
f1 = f1_score(y_test, test_pred, average='weighted')

print(f"Test Precision: {precision:.4f}")
print(f"Test Recall: {recall:.4f}")
print(f"Test F1 score: {f1:.4f}")
```

```
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent worke
rs.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.0s
[Parallel(n_jobs=8)]: Done 184 tasks      | elapsed:    0.2s
[Parallel(n_jobs=8)]: Done 434 tasks      | elapsed:    0.4s
[Parallel(n_jobs=8)]: Done 784 tasks      | elapsed:    0.8s
Test Precision: 0.4850
Test Recall: 0.4888
Test F1 score: 0.4839
[Parallel(n_jobs=8)]: Done 1000 out of 1000 | elapsed:    1.0s finished
```