Machine Learning

Assignment #1 (Supervised Learning)

Prof. Eunwoo Kim

Submit all codes written in Python to run the following algorithms and a pdf file that describes your experimental setup and results.

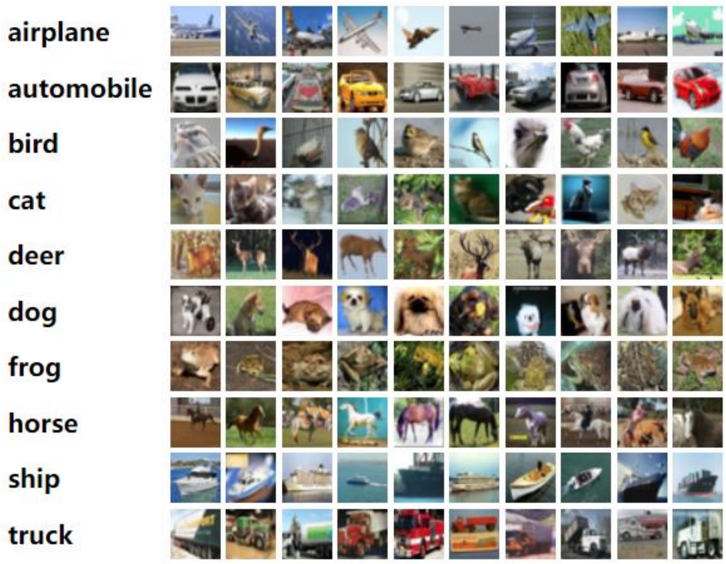
Algorithms to be implemented and learned:

- ✓ Decision Tree and Support Vector Machine (SVM)

Datasets:

- ✓ MNIST (10 classes, 60000 training images and 10000 test images)

- ✓ CIFAR-10 (10 classes, 50000 training images and 10000 test images)



What you have to submit (through e-class)

- ✓ Implementation codes (zip file) written in Python for the algorithms.
- ✓ A report (pdf file) describing experimental setup and results on train/test sets.

You can follow the guideline on the experimental setup with the PyTorch library we provide:

- First, import the packages needed for learning.

- About pytorch : [torch · PyPI]
- About numpy : [numpy · PyPI]

```
from torch.utils.data import DataLoader
from torchvision import transforms, datasets
import numpy as np
from sklearn import svm
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from scipy.stats import randint
from sklearn.model_selection import GridSearchCV
```

- The next step involves loading the data will be used for learning.

- CIFAR-10 dataset: https://www.kaggle.com/competitions/cifar-10/data
- MNIST dataset: https://www.kaggle.com/datasets/oddrationale/mnist-in-csv

- It extracts the input images and their labels from the dataloaders and concatenates them into numpy arrays, CIFAR_train_images, CIFAR_train_labels, CIFAR_test_images, and CIFAR_test_labels. The images are flattened into vectors and normalized to have values between 0 and 1. These numpy arrays can be used for training and testing machine learning models.

```
# CIFAR-10
CIFAR_transform_train = transforms.Compose([transforms.ToTensor()])
CIFAR_transform_test =transforms.Compose([transforms.ToTensor()])

trainset_CIFAR = datasets.CIFAR10(root='./data', train=True, download=True, transform=
CIFAR_transform_train)
testset_CIFAR = datasets.CIFAR10(root='./data', train=False, download=True,
transform=CIFAR_transform_test)

CIFAR_train = DataLoader(trainset_CIFAR, batch_size=32, shuffle=True, num_workers=2)
CIFAR_test = DataLoader(testset_CIFAR, batch_size=32, shuffle=False, num_workers=2)

CIFAR_train_images = []
CIFAR_train_labels = []
for batch in CIFAR_train:
    images, labels = batch
    images_flat = images.view(images.shape[0], -1)
    CIFAR_train_images.append(images_flat.numpy())
    CIFAR_train_labels.append(labels.numpy())
```

```
CIFAR_train_images = np.vstack(CIFAR_train_images)
CIFAR_train_labels = np.concatenate(CIFAR_train_labels)

CIFAR_test_images = []
CIFAR_test_labels = []
for batch in CIFAR_test:
    images, labels = batch
    images_flat = images.view(images.shape[0], -1)
    CIFAR_test_images.append(images_flat.numpy())
    CIFAR_test_labels.append(labels.numpy())
CIFAR_test_images = np.vstack(CIFAR_test_images)
CIFAR_test_labels = np.concatenate(CIFAR_test_labels)
```

- The MNIST dataset can be obtained in a similar way that the CIFAR-10 dataset is collected.

```
# MNIST
mnist_train_transform =  transforms.Compose([transforms.ToTensor()])
mnist_test_transform =  transforms.Compose([transforms.ToTensor()])

trainset_mnist = datasets.MNIST(root='./data', train=True, download=True, transform=mnist_train_transform)
testset_mnist = datasets.MNIST(root='./data', train=False, download=True, transform=mnist_test_transform)

MNIST_train = DataLoader(trainset_mnist, batch_size=32, shuffle=True, num_workers=2)
MNIST_test = DataLoader(testset_mnist, batch_size=32, shuffle=False, num_workers=2)

MNIST_train_images = []
MNIST_train_labels = []
for batch in MNIST_train:
    images, labels = batch
    images_flat = images.view(images.shape[0], -1)
    MNIST_train_images.append(images_flat.numpy())
    MNIST_train_labels.append(labels.numpy())
MNIST_train_images = np.vstack(MNIST_train_images)
MNIST_train_labels = np.concatenate(MNIST_train_labels)

MNIST_test_images = []
MNIST_test_labels = []
for batch in MNIST_test:
    images, labels = batch
    images_flat = images.view(images.shape[0], -1)
    MNIST_test_images.append(images_flat.numpy())
    MNIST_test_labels.append(labels.numpy())
MNIST_test_images = np.vstack(MNIST_test_images)
MNIST_test_labels = np.concatenate(MNIST_test_labels)
```

What you have to do:

- Implement **Decision Tree** (you should manually set some hyperparameters involved).
  - You may use DecisionTreeClassifier in the library or you can write your own one.
    - Each pixel is regarded as an attribute to construct a node.
    - Tree depth: 3, 6, 9, 12 (show all results with four different depths)
    - Use GridSearchCV to search proper hyperparameters (set cross validation k to 5).
    - Define the search pool for GridSearchCV as follows. (Note that the definition of the search pool can vary depending on the user's preferences or requirements.)

```
params_grid = { 'min_samples_split': [2, 5, 10],
                'min_samples_leaf': [1, 2, 4],
                'max_leaf_nodes': [5, 10, None]}
```

  - Learn decision tree on training examples and run it for test examples.
  - Show accuracy on training and test sets in a table.
- Implement **SVM** in a similar way to Decision Tree.
  - You may use svm.svc for classification.
  - Use linear and kernel SVMs (you can set kernel = 'linear' and 'rbf').
  - Show accuracy on training and test sets in a table.
- It's expected that the codes will be performed in minutes to hours depending on the data under the Colab environment.

Inquiry (via email)

- Class-01: Jungkyoo Shin (neo293@cau.ac.kr)
- Class-02: Jiho Lee (j2hoooo@cau.ac.kr)