Appendix-4: SVM without Kernel

Support Vector Machines (SVM) are a powerful classification algorithm that learns a decision boundary by maximizing the margin between classes. It can be extended to multiclass classification using one-vs-one or one-vs-rest schemes.

We used scikit-learn's SVM implementation without kernel (linear) with default hyperparameters in this notebook.

In [1]:
```
# import library dependencies
import numpy as np
from sklearn import svm
from sklearn.metrics import accuracy_score, precision_score, recall_score, f
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 and scale the pixel values to [0, 1]
x_train = x_train.reshape((x_train.shape[0], -1))
x_test = x_test.reshape((x_test.shape[0], -1))
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
```

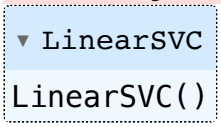
Define model and train

In [12]:
```
# define a linear SVM classifier
svm_wk = svm.LinearSVC()
```

In [13]:
```
# train the classifier on the training set
svm_wk.fit(x_train, y_train.ravel())
```

```
/Users/shrivastavasatyam/miniconda3/lib/python3.10/site-packages/sklearn/svm
/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase t
he number of iterations.
  warnings.warn(
```

Out[13]:



Save and load model

In [14]:
```
# Save the model to a file
joblib.dump(svm_wk, 'svm_without_kernel.sav')
```

Out[14]:
```
['svm_without_kernel.sav']
```

In [15]:
```
# Load the saved model from a file
loaded_model = joblib.load('svm_without_kernel.sav')
```

In [16]:
```
# make predictions on the test set
y_pred = loaded_model.predict(x_test)
```

Evaluate the model

In [17]:
```
# compute the accuracy, precision, recall, and F1 score of the classifier
train_acc = accuracy_score(y_train.ravel(), loaded_model.predict(x_train))
test_acc = accuracy_score(y_test.ravel(), y_pred)
precision = precision_score(y_test.ravel(), y_pred, average='macro')
recall = recall_score(y_test.ravel(), y_pred, average='macro')
f1 = f1_score(y_test.ravel(), y_pred, average='macro')
```

In [18]:
```
# print accuracy, precision, recall, and F1 score
print('Train Accuracy:', train_acc)
print('Test Accuracy:', test_acc)
print('Precision: {:.3f}'.format(precision))
print('Recall: {:.3f}'.format(recall))
print('F1 Score: {:.3f}'.format(f1))
```

```
Train Accuracy: 0.27592
Test Accuracy: 0.2202
Precision: 0.420
Recall: 0.220
F1 Score: 0.170
```