LAHORE UNIVERSITY OF MANAGEMENT SCIENCES
Syed Babar Ali School of Science and Engineering

EE212 Mathematical Foundations for Machine Learning and Data Science
Summer Semester 2020

Laboratory 5 – Application of Supervised Learning

Issued: Sunday 09 August, 2020

Total Marks: 100

Contribution to Final Assessment: 2%

Submission: 12:15 pm, Tuesday 11 August, 2020.

Goal

The goal of this laboratory is to learn different techniques used in supervised learning. The category of supervised learning that we will be dealing with in this lab is called a classification problem. We will train and test our classifier on MNIST data set.

Instructions

If you have any concerns, you can ask us in the live zoom session, or in the chat. Each of you has been allotted a TA/RA, so when you are done with the lab, let them know and they will mark it. It is your responsibility to ensure you get your work checked.

Name your files Task1.py, Task2.py and so on. Compress them in a **single** file and name it as LabXX_YourRollNumber. Submit this file on LMS before the deadline. No late submissions will be accepted.

Only those uploaded tasks will be marked which has been shown to the TA/RA and marked during the live session.

Before starting, import the following libraries from python:

```
import numpy as np
from mnist import MNIST
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVC
from sklearn import metrics
from skimage.feature import hog
from sklearn.decomposition import PCA
```

Task 1: SVM on raw data (30 Marks)

Raw Data

The data that we will be using in this lab is MNIST data. This is a collection of images of handwritten digits, 0-9, along with their labels. The purpose of these tasks will be to identify the digit in each image and predict its label.

Classifier

In supervised learning, the data is split into two parts; training data and testing data. We use the training data along with its labels, to train our model. For testing, we only input the testing data and not the labels. We use our trained model to predict the labels of this testing data and compare it with the actual labels to see how accurate our model is.

The classifier that we will be using is multiclass **Support Vector Machine (SVM)**. Without getting into much detail, SVM tries to learn the boundary between different classes. You can imagine data as a scatter plot, with data points of each class lying together as a cluster in a 2D plane. Now imagine drawing a boundary such the boundary separates/isolates each class from each other in the plane. SVM tries to learn that boundary and use it to classify data.

Use the following line of code to load the training and testing data. Download the 4 files uploaded on LMS and save them in your local directory. Do not change their names. Use the following code to load the data into relevant matrices (replace 'directory-path' with your local path where the files are saved):

```
mndata = MNIST('directory-path')

x_train, y_train = mndata.load_training()
x_test, y_test = mndata.load_testing()

X_train=np.array(x_train)
y_train=np.array(y_train)
X_test=np.array(x_test)
y_test=np.array(y_test)
```

You should have **60,000** training images, each with dimensions **1x784**, and labels. The testing data should contain **10,000** images and labels. Check it after you have loaded the data.

1. We will now scale the data so that each value is between -1 and 1. Use the following code for scaling your data:

   ```
   scaling = MinMaxScaler(feature_range=(-1, 1)).fit(x_train)
   x_train = scaling.transform(x_train)
   x_test = scaling.transform(x_test)
   ```

2. Use multiclass SVM from the sklearn library. Use a **linear kernel** and train the model using training data. After training it, use the model to predict labels for the testing data.

Once we have the predicted labels, we can make a confusion matrix. It is a matrix with predicted labels on one axis and actual labels on the other. The number in cell indicates the number of times the corresponding actual label was classified as the corresponding predicted label. The diagonal of course represents the correct predictions while the off diagonal terms are wrong predictions.

3. Using the in built function, **'metrics'**, find the confusion matrix of actual and predicted labels of the testing data. You can plot it using the following code (cm is the confusion matrix):

```
   plt.figure()
plt.imshow(cm, interpolation='nearest', cmap='Pastel1')
plt.title('Confusion matrix', size = 15)
plt.colorbar()
tick_marks = np.arange(10)
plt.xticks(tick_marks,
  ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"],
rotation=45, size = 10)
plt.yticks(tick_marks,
  ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"],
  size = 10)
plt.tight_layout()
plt.ylabel('Actual label', size = 15)
plt.xlabel('Predicted label', size = 15)
width, height = cm.shape

for x in range(width):
    for y in range(height):
        plt.annotate(str(cm[x][y]), xy=(y, x),
                     horizontalalignment='center',
                     verticalalignment='center')

plt.show()
```

Using the confusion matrix, we can calculate different merits of our model. Two such merits are **Accuracy** and **False Positive Rate (FPR)** :

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

which can be calculated from the confusion matrix as:

$$\text{Accuracy} = \frac{\text{Sum of diagonal of the matrix}}{\text{Sum of all values in the matrix}}$$

FPR for each class corresponds to the probability of a false alarm. It is when the classifier detects the data as a specific class but the data does not belong to that class. Each class will have its own FPR

$$\text{FPR}_i = \frac{\text{Total number of wrong predictions of class } i}{\text{Total number of predictions of class } i}$$

which can be calculated from the confusion matrix as:

$$\text{FPR}_i = \frac{\text{Sum over the row of true label } i \text{ except the diagonal}}{\text{Sum over the row of true label } i}$$

4. Use the confusion matrix to calculate FPR for each class and overall accuracy of the model.

Task 2: Multiclass SVM with PCA (35 marks)

In the previous task, we used all 784 pixels in the training of our model due to which we have very high model complexity (number of parameters or dimension). In this task we will use Principal Component Analysis (PCA) technique to reduce the dimensionality of the data. As we have studied the course, PCA is a linear dimensionality reduction technique that embeds higher dimensionality data into a lower dimensionality subspace. This is enabled by linear transformation to retain the principal components which account for most of the variation in the original higher dimensional data.

1. Load the MNIST data as you did in Task 1.
2. Use the following code to extract the principle components of data and then projects your image along those components. The variable *n_components* determines how many components you wish to consider.

```
n_components=2
pca=PCA(n_components)

# Transform data
pca.fit(x_train)
x_train=pca.transform(x_train)
x_test=pca.transform(x_test)
```

   P.S. Look at the shape of training and testing data now. The second dimension corresponds to the number of components.

3. Scale the data and use SVM to predict labels as you did in Task 1.
4. Compute the confusion matrix along with the accuracy and FPR for each class.

   Your accuracy might be less than the one you got while using raw data. This is because perhaps the number of components you are using are not enough.

5. Perform this analysis again but for the following number of components and compute the accuracy in each case:
   - $components = 5$
   - $components = 11$
   - $components = 44$

   P.S. You can compute the accuracy using the command **metrics.accuracy_score(y_test, y_predict)** *where y_predict are the predicted labels.*

6. Comment on the time taken for classification using PCA as compared to using raw data. Why do you think there is a difference? Why does increasing the number of components leads to improved accuracy?

Task 3: Multiclass SVM with Histogram of Oriented Gradient (HOG) Features (35 marks)

In this Task, we will classify the digits by first computing using histogram of oriented gradient (HOG) features and then using a multiclass SVM classifier on the HOG features. This is to illustrate that the utilizing useful features of the data can help us in improving the accuracy of the model. The HOG feature is widely used in machine learning and image processing for image classification and object detection tasks. Computing HOG feature provides the counts occurrences of gradient orientation in localized portions of an image. For handwritten digits, HOG has been shown to be efficient feature descriptor due to its robustness to the variation in the data for each class, we expect the improvement in the classification accuracy.

1. Load the MNIST data as you did in Task 1.
2. Use the following code to extract HOG features of data, using the function calc_hog that we define ourselves:

```
# Function that computes HOG features
def calc_hog(X, image_shape=(28, 28), pixels_per_cell=(10, 10)):
    fd_list = []

    # Loop over all images
    for row in X:
        img = row.reshape(image_shape)
        fd = hog(img, orientations=10, pixels_per_cell=pixels_per_cell,
                    cells_per_block=(1, 1))
        fd_list.append(fd)

    return np.array(fd_list)

# Now data is not raw images but HOG features
x_train = calc_hog(x_train)
x_test = calc_hog(x_test)
```

P.S. Look at the shape of training and testing data now. The second dimension corresponds to the number of features extracted.

3. Scale the data and use SVM to predict labels as you did in Task 1.
4. Compute the confusion matrix along with the accuracy and FPR for each class.

   Your accuracy might be less than the one you got while using raw data. This is because perhaps the features you are using are not enough.

5. Tweak the value of n in *pixels_per_cell=(n,n)* in the calc_hog function and in *orientations=n* in the hog function to increase your features. Increase them enough to get better accuracy than when you used raw data.
6. Comment on the time taken for classification using HOG features as compared to using raw data. Why do you think there is a difference? Is there a trade-off between speed and accuracy?