DRISHYA UNIYAL	MT21119

A. Coding.

Dataset: MNIST

Assumptions and Additions:

1. Initially, all the matrices and arrays are appended with zeros.
2. The code is written from scratch, and a few extra things have been added.
3. Formulas used are the ones explained in class.

To handle a few cases:

1. Add regularization term.
2. Check determinant should not be zero. Add values to diagonals there.
3. Get the results.

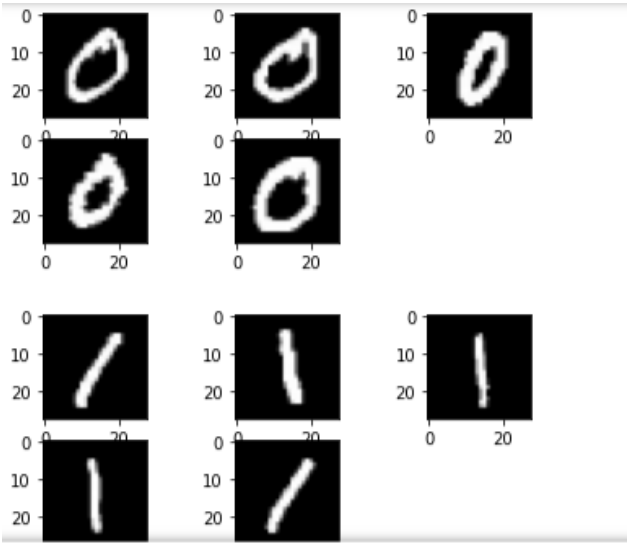
(a) Download the dataset, and visualize 5 samples from each class in the images.

Solution:

Steps:

1. Import the libraries.
2. Load the Dataset given.
3. Use for loops to get the image classes in range (0,10) as we have ten classes.
4. Loop within the above loop to get 5 random samples from each class.
5. Use matplotlib to plot the results.

Results:



Similarly, the rest of the classes are plotted.

(b) Implement Linear Discriminant analysis(LDA) from scratch.

(i) Compute a covariance matrix using a weighted average of covariance of each Class, and let's call it Σg.

S_W is the within-class covariance matrix equation.

x_n are the data points, m_1 is the mean of class, 1 and m_2 is the mean of class // W

$$\mathbf{S}_{\mathrm{W}} = \sum_{n \in \mathcal{C}_1} (\mathbf{x}_n - \mathbf{m}_1)(\mathbf{x}_n - \mathbf{m}_1)^{\mathrm{T}} + \sum_{n \in \mathcal{C}_2} (\mathbf{x}_n - \mathbf{m}_2)(\mathbf{x}_n - \mathbf{m}_2)^{\mathrm{T}}$$

(ii) Assume Σg to be the covariance for all classes while implementing the Linear Discriminant function. (Hint: recall the discriminant function with assumption of same covariance matrix for each class)

```
for c in self.classes:
    X_c = X[y == c]
    self.priors[c] = X_c.shape[0] / float(n_samples) #calculate priors
    class_means.append(X_c.mean(axis=0))  #calculate means for the classes
    cov = np.zeros((self.D, self.D))  #initially the matrix is zero
    for i in range(X_c.shape[0]):
        row = X_c[i, :].reshape(self.D, 1)
        cov += np.dot(row, row.T)
    cov /= X_c.shape[0]
    cov += np.eye(self.D) * self.reg_covar
    class_covs.append(cov)
    class_covs_inv.append(np.linalg.inv(cov))

self.class_mean = np.array(class_means)
self.covariance = class_covs
self.covariance_inv = class_covs_inv
```

Solution for b(i) and b(ii)

Steps:

1. Load all the necessary libraries.
2. Load the dataset.
3. Initially, all the matrix and arrays are zero.
4. Calculate the class means.
5. Calculate the covariance matrix using the formula. (use of determinant and inverse from numpy for this calculation).
6. Loop through each class and calculate the log-likelihood.

```
log_likelihood[j, i] = -0.5 * (np.dot(np.dot((row - mean), cov_inv), (row - mean).T) + log_det) + np.log(self.priors[i])
#calculate the log likelihood
```

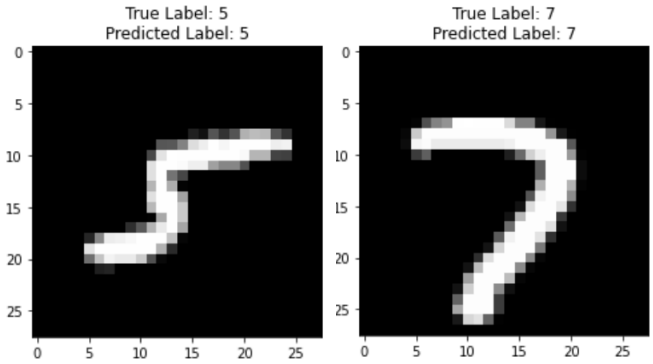
(c) Use LDA parameters calculated from training data to predict the class of samples from testing data.

```
lda = LDA()
lda.fit(train_data, train_labels)
y_pred_lda = lda.predict(test_data)
acc = lda.accuracy(test_data, test_labels)
print("Accuracy for LDA",acc)
```

Steps:

1. Call the LDA class made.
2. Fit the class with X_train and y_train.
3. Predict the samples from the test data.
4. Get Results.

Results:



(d) Implement the Quadratic discriminant analysis(QDA) from scratch. Follow these instructions:

(i) You will need to calculate the covariance matrix of each class separately and they will be different.

(ii) Implement Quadratic discriminant function.

Solution for d(i) and d(ii)

```
for c in self.classes:
    X_c = X[y == c]
    mean_c = np.mean(X_c, axis=0)
    n_samples = X_c.shape[0]
    X_centered = X_c - mean_c
    cov_c = np.dot(X_centered.T, X_centered) / (n_samples - 1)
    self.mean_vectors.append(mean_c)
    cov_c_regularized = cov_c + self.reg_param * np.eye(X.shape[1])
    self.cov_matrices.append(cov_c_regularized)
    self.priors.append(X_c.shape[0] / X.shape[0])
return self
```

Steps:

1. Almost the same approach as LDA.
2. Calculate the priors.
3. Calculate the means.
4. Calculate the covariance for each class that is going to be different.
5. Calculate Log Likelihood.

```
log_probs[:, i] = -0.5 * np.sum(np.dot(X_centered, cov_c_inv) * X_centered, axis=1) - \
                  0.5 * np.log(cov_c_det) + np.log(self.priors[i])
```

6. Predict and get the results.

(e) Use QDA parameters calculated from training data to predict the class of samples from testing data.

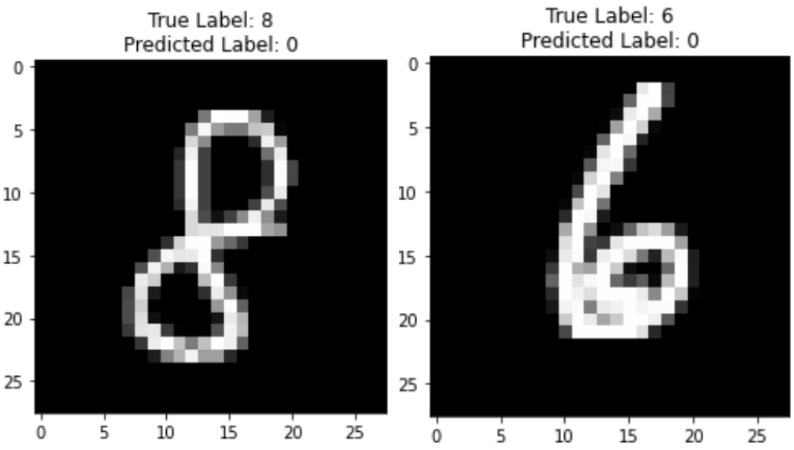
```
qda = QDA(reg_param=0.1)
qda.fit(train_data,train_labels)
y_pred_qda = qda.predict(test_data)
acc = qda.acc(test_data, test_labels)
print("Accuracy For QDA",acc)
```

Steps:

1. Call the class QDA.
2. Fit it.
3. Predict for test samples.
4. Print the results.

Results:

This can be looped for more predictions as in code!



(f) Implement an accuracy function (Not allowed to use the inbuilt library) and report accuracy on testing data for both LDA and QDA.

Steps:

1. The accuracy function is implemented from scratch.

```
def accuracy(self, X, y):
    y_pred = self.predict(X)
    return np.mean(y_pred == y)
```

Predicted and True Labels

The predict function is also made in the LDA and QDA classes.

(g) Use LDA and QDA functions of the sklearn library and compare the results with your implementation.

Steps:

1. Use the LinearDiscriminantAnalysis, and QuadraticDiscriminantAnalysis from the sklearn library and get the results.
2. Call the class.
3. Fit the method.
4. Predict.
5. Get Accuracy. (here, accuracy is from the sklearn).

Results.

	Sklearn	**Scratch**
LDA	87.3	77.98
QDA	54.05	9.8

Interpretations:

1. LDA still receives better accuracy but QDA performed bad.
2. Reason: Not too optimized for better performance, case missing.