ECE 661 Computer Vision: HW11

Qiuchen Zhai
qzhai@purdue.edu

December 3, 2020

1 Task 1: Face Recognition

1.1 PCA: Principal Component Analysis

The PCA improves the calculation efficiency by representing data in high-dimension by low-dimension features. Given the training dataset, the following steps are implemented for PCA feature representation,

- Vectorize the training dataset and normalize the dataset $\vec{x}_i, i = 1, 2, ..., N$.
- Compute the mean image vector

$$\vec{m} = \frac{1}{N}\sum_i \vec{x}_i \tag{1}$$

- Estimate the covariance of the image set

$$C = \frac{1}{N}\sum_{i=1}^{N}(\vec{x}_i - \vec{m})(\vec{x}_i - \vec{m})^T = \frac{1}{N}XX^T \tag{2}$$

- Calculate the eigen-values and eigen-vectors w_k of XX^T and form the orthogonal PCA feature set by

$$W_K = [\vec{w}_1|\vec{w}_2|...|\vec{w}_K] \tag{3}$$

  where the K eigen-vectors correspond to the K largest eigenvalues.
- Dimensionality reduction by PCA.

$$y = \vec{W}^T(x - \vec{m}) \tag{4}$$

1.2 LDA: Linear Discriminant Analysis

The between class scatter and winthin class scatter are defined for multiple classes as

$$S_B = \frac{1}{|C|}\sum_{i=1}^{|C|}(\vec{m}_i - \vec{m})(\vec{m}_i - \vec{m})^T \tag{5}$$

$$S_W = \frac{1}{|C|}\sum_{i=1}^{|C|}\frac{1}{|C_i|}\sum_{k=1}^{|C_i|}(\vec{x}_k^i - \vec{m}_i)(\vec{x}_k^i - \vec{m}_i)^T \tag{6}$$

We would like to search for the directions that maximize the ratio of between-class scatter to within-class scatter, which is known as the Fisher Discriminant Function $J(\vec{w})$

$$\vec{w} = \arg\max J(\vec{w}) = \arg\max \frac{\vec{w}^T S_B \vec{w}}{\vec{w}^T S_w \vec{w}} \tag{7}$$

Next, we compute the $\vec{w}$ following the Yu and Yang's algorithm:

- Vectorize the training dataset and normalize the dataset.
- Compute the global mean and class mean of training dataset:

$$\vec{m} = \frac{1}{N} \sum_k \vec{x}_k \tag{8}$$

$$\vec{m}_i = \frac{1}{|C|} \sum_{k=1}^{|C|} \vec{x}_k \tag{9}$$

  where the cardinality $|C|$ gives the number of all classes.

- Calculate the mean matrix M.

$$M = [\vec{m}_1 - m|\vec{m}_2 - m|...|\vec{m}_i - m|...|\vec{m}_{|C|} - m] \tag{10}$$

- Calculate the eigen-vectors $\vec{u}$ of $M^T M$ and then compute the eigen-vectors of S_B by $\vec{V} = M\vec{u}$.
- Construct matrix $Z = Y D_B^{(-1/2)}$ where $Y = [\vec{V}_1|\vec{V}_2|...|\vec{V}_{|C|}]$ and D_B is the eigen-value of S_B.
- Compute the eigen-vectors U of $Z^T S_W Z$ through $Z^T S_W Z = (Z^T X_W)(Z^T X_W)^T$, where $X_W = [\vec{x}_{11} - \vec{m}_1|\vec{x}_{12} - \vec{m}_1|...|\vec{x}_{|C_1|} - \vec{m}_1|...|\vec{m}_{|C_i|} - \vec{m}_i]$.
- Compute the matrix of the LDA eigenvectors

$$W^T = \hat{U}^T Z^T \tag{11}$$

- Project the image to the subspace before using.

$$y = \vec{W}^T (x - \vec{m}) \tag{12}$$

1.3 Classification Accuracy

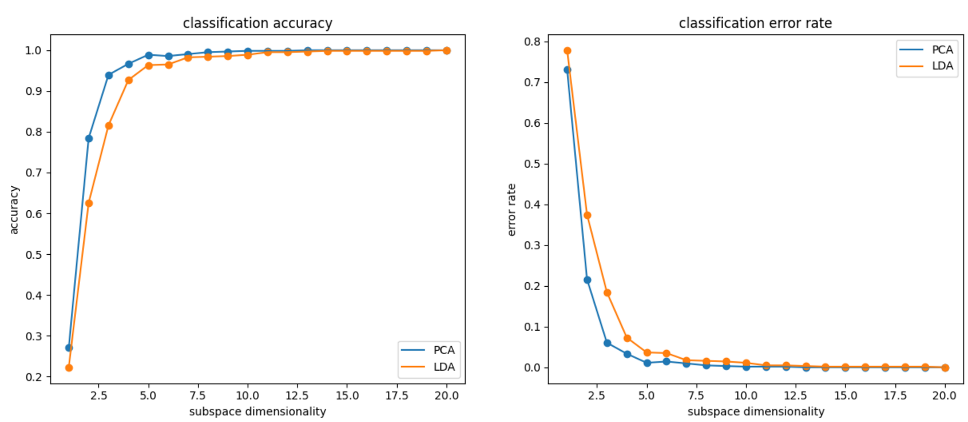

Figure 1: Classification accuracy and error rate

1.4 Comparison of the results for PCA and LDA

From the plot, we could see that both classification algorithms has more accurate classification with larger subspace dimensionality. However, LDA needs larger subspace dimensionality (K>19) to achieve 100% accuracy while PCA only needs K>12 to realize 100% classification accuracy. Overall, PCA has better performance in terms of classification accuracy especially when the dimensionality of subspace is low.

2 Task 2: Object Detection with Cascaded AdaBoost Classfication

2.1 Outline of the classifier

The following subsections described the procedure required for object detection with cascaded adaboost classification.

2.1.1 Haar-like feature extraction

To build the weak classifiers, we extract the Haar-like feature from the train dataset and threshold the feature. The simplest horizontal and vertical Haar kernel is given by $[0, 1]$ and $[0, 1]^T$. In this experiment, each image in our data set is of size 20 x 40. For computation efficiency, we implement the horizontal Haar filters of size 1 x 2, 1 x 4, ..., 1 x 40 and vertical Haar filters of size 2 x 2, 4 x 2, ..., 20 x 2. The Haar filters will slide over the integral image along with corresponding axis. Then we will get the returned feature values.

2.1.2 Weak Classifiers

To build the final strong classifiers, we first build a total number of T weak classifiers. Then we select the best weak classifiers iteratively. To find the best weak classifiers, we first initialize the weights of samples with equal distribution. Suppose we have p positive training samples and n negative training sample, then the initial weight for positive and negative samples are $\frac{1}{2p}$ and $\frac{1}{2n}$ separately. Note that we need to normalize the weight before using them. Then we will evaluate the weak classifiers by thresholding the features. The threshold is calculated by finding the minimum error given by

$$e = \min(S^+ + (T^- - S^-), S^- + (T^+ - S^+)) \tag{13}$$

where the $+$ refers to positive samples and $-$ refers to negative samples, S denotes the sum of weights below the current threshold value and T denotes the total sum of weights corresponding example. Let h_t denote each weak classifier at t^{th} iteration. The weak classifier that minimizes the error is regarded to be the best weak classifier $h_t(x, f_t, p_t, \theta_t)$ where f_t denotes the feature value, p_t denotes the polarity and θ_t denotes the threshold value at t^{th} iteration. For each iteration, we then calculate $\beta_t = \frac{e_t}{1-et}$ the confidence factor $\alpha_t = \log \frac{1}{\beta_t}$. The current best weak classifiers will be used to constitute the current version of strong classifier. And we update the weights by

$$w_{t+1,i} = w_{t,i}\beta_t^{1-e_i} \tag{14}$$

where e_i is the indicator to show that whether the sample is classified corrrectly or not. In other words, $e_i = 1$ if the sample is NOT classified correctly and $e_i = 0$ if the sample is classified correctly. The iteration continues until we obtain enough number of best weak classifiers or any stop criteria is reached.

2.1.3 Build Strong Classifier by Cascaded Adaboost

After obtaining the best weak classifiers, then the strong classifier is constituted by

$$C(x) = \begin{cases} 1, & \sum \alpha_t h_t(x) \geq \frac{1}{2} \sum \alpha_t, \\ 0, & otherwise. \end{cases} \tag{15}$$

The procedure of constructing one strong classifier will be repeated for integrating adaboost with cascaded algorithm. For each stage, only the negative images which are correctly recognized and all the positive images will pass through. In each stage, the false positive rate and the false negative rate are computed by

$$FP\ rate = \frac{\#\ of\ misclassified\ negative\ test\ images}{Total\ \#\ of\ negative\ test\ images} \tag{16}$$

$$FN\ rate = \frac{\#\ of\ misclassified\ positive\ test\ images}{Total\ \#\ of\ positive\ test\ images} \tag{17}$$

If the false positive rate is smaller than acceptable rate, then we think the stage is good enough for completion.

2.2 Testing and Results

The classifier is yet to be tested.

3 Appendix

3.1 Source Code of Task1

```python
import numpy as np
import cv2
import os
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier


# read images
def load_img(img_path):
    train_dataset = []
    test_dataset = []
    # read training data
    train_folder = img_path + "train"
    train_fseq = os.listdir(train_folder)
    train_fseq.sort()
    for fname in train_fseq:
        img = cv2.imread(os.path.join(train_folder, fname))
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        vec = gray.flatten()                                            # 16348x1
        train_dataset.append(vec)
    train_dataset = np.asarray(train_dataset).T                         # (16384, 630)
    train_dataset = train_dataset / np.linalg.norm(train_dataset, axis=0)
    mean_vec = np.expand_dims(np.mean(train_dataset, axis=1), axis=1)
    train_dataset = train_dataset - mean_vec
    # read testing data
    test_folder = img_path + "test"
    test_fseq = os.listdir(test_folder)
    test_fseq.sort()
    for fname in test_fseq:
        img = cv2.imread(os.path.join(test_folder, fname))
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        vec = gray.flatten()
        test_dataset.append(vec)
    test_dataset = np.asarray(test_dataset).T                            # (16384,
    630)
    test_dataset = test_dataset / np.linalg.norm(test_dataset, axis=0)
    test_dataset = test_dataset - mean_vec
    # create labels
    train_label = []
    for i in range(30):
        train_label.append([i + 1] * 21)
    train_label = np.asarray(train_label).flatten()
    test_label = train_label

    return train_dataset, test_dataset, train_label, test_label, mean_vec


img_path = "ECE661_2020_hw11_DB1/"
train_data, test_data, train_label, test_label, global_mean = load_img(img_path)
# print(train_data.shape, test_data.shape, global_mean.shape)


def PCA_feature_extraction(data, num_eig):
    u, s, vh = np.linalg.svd(np.matmul(data.T, data))     # 630 x  630
    w = np.matmul(data, vh.T)
    w = w / np.linalg.norm(w, axis=0)
    w = - w[:, 0:num_eig]                                # 16348 x K
    PCA_val = np.matmul(w.T, data)

    return w, PCA_val
```

```
def NN_classifier(train_feature, test_feature, train_label, test_label):
    classifier = KNeighborsClassifier(n_neighbors=1)
    classifier.fit(train_feature.T, train_label)
    predict_label = classifier.predict(test_feature.T)
    accuracy_mat = np.zeros(test_label.shape)
    accuracy_mat[predict_label == test_label] = 1
    accuracy = np.sum(accuracy_mat) / 630

    return accuracy


def calculated_Z(M):
    # compute LDA eigen vecs
    u, s, vh = np.linalg.svd(np.matmul(M.T, M))
    # print(s.shape)                                      # (30,)
    evecs = np.matmul(M, vh.T)                            # (16384, 30)
    # construct z matrix
    Z = np.matmul(evecs, np.diag(s ** (-0.5)))

    return Z


class_mean = np.zeros((16384, 30))
mean_mat = []
for i in range(30):
    class_mean[:, i] = np.mean(train_data[:, i*21:(i+1)*21] + global_mean, axis=1)
mean_mat = class_mean - global_mean
Z = calculated_Z(mean_mat)


# diagonalize Z
X_w = np.zeros(train_data.shape)
for i in range(30):
    X_w[:, i*21:(i+1)*21] = train_data[:, i*21:(i+1)*21] + global_mean - np.
    expand_dims(mean_mat[:, i], axis=1)


def LDA_feature_extraction(Z, X_w, num_eig):
    temp = np.matmul(Z.T, X_w)
    u, s, vh = np.linalg.svd(np.matmul(temp, temp.T))
    w = np.matmul(Z, vh.T)
    w = w / np.linalg.norm(w, axis=0)
    w = w[:, 0:num_eig]
    return w                                # LDA feature


PCA_acc_ls = []
PCA_err_ls = []
LDA_acc_ls = []
LDA_err_ls = []
for k in range(20):
    K = k + 1
    # PCA
    train_PCAfeature, train_PCAval = PCA_feature_extraction(train_data, num_eig=K)
    test_PCAval = np.matmul(train_PCAfeature.T, test_data)
    PCA_accuracy = NN_classifier(train_PCAval, test_PCAval, train_label, test_label)
    PCA_acc_ls.append(PCA_accuracy)
    PCA_err_ls.append(1 - PCA_accuracy)
    # LDA
    w = LDA_feature_extraction(Z, X_w, num_eig=K)
    train_LDAval = np.matmul(w.T, train_data)
    test_LDAval = np.matmul(w.T, test_data)
```

```
    LDA_accuracy = NN_classifier(train_LDAval, test_LDAval, train_label, test_label)
    LDA_acc_ls.append(LDA_accuracy)
    LDA_err_ls.append(1 - LDA_accuracy)


# plot
M = [x+1 for x in range(20)]
plt.subplot(121)
plt.scatter(M, PCA_acc_ls)
plt.plot(M, PCA_acc_ls)
plt.scatter(M, LDA_acc_ls)
plt.plot(M, LDA_acc_ls)
plt.legend(['PCA', 'LDA'])
plt.ylabel('accuracy')
plt.xlabel('subspace dimensionality')
plt.title('classification accuracy')
plt.subplot(122)
plt.scatter(M, PCA_err_ls)
plt.plot(M, PCA_err_ls)
plt.scatter(M, LDA_err_ls)
plt.plot(M, LDA_err_ls)
plt.legend(['PCA', 'LDA'])
plt.ylabel('error rate')
plt.xlabel('subspace dimensionality')
plt.title('classification error rate')
plt.show()
print(M)
print(PCA_acc_ls)
print(LDA_err_ls)
```

Task1.py

3.2 Source Code of Task2

```
import numpy as np
import os
import cv2
import math


# ================================ read dataset ========================================
def load_img(img_path):
    pos_dataset = []
    pos_folder = img_path + "positive"
    pos_fseq = os.listdir(pos_folder)
    pos_fseq.sort()
    for fname in pos_fseq:
        img = cv2.imread(os.path.join(pos_folder, fname))
        if img is not None:
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            pos_dataset.append(gray)
    pos_dataset = np.asarray(pos_dataset)                          # (710, 20, 40)
    pos_label = np.asarray([[1]*len(pos_fseq)])                    # (1, 710)

    neg_dataset = []
    neg_folder = img_path + "negative"
    neg_fseq = os.listdir(pos_folder)
    neg_fseq.sort()
    for fname in neg_fseq:
        img = cv2.imread(os.path.join(neg_folder, fname))
        if img is not None:
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```python
            neg_dataset.append(gray)
    neg_dataset = np.asarray(neg_dataset)                          # (710, 20, 40)
    neg_label = np.asarray([[0] * len(neg_fseq)])                  # (1, 710)

    return pos_dataset, pos_label, neg_dataset, neg_label


train_path = "ECE661_2020_hw11_DB2/train/"
train_pos_data, train_pos_label, train_neg_data, train_neg_label = load_img(train_path)
# print(train_pos_data.shape, train_pos_label.shape, train_neg_data.shape, train_neg_label.shape)


# =============================== compute feature ================================
def uint(num):
    return np.uint8(num)


def sum_pixels(img, A, B, C, D):
    output = img[uint(D[0]), uint(D[1])] - img[uint(B[0]), uint(B[1])] - img[uint(C[0]), uint(C[1])] + img[uint(A[0]), uint(A[1])]
    return output.astype(np.float64)


def Haar_feature_extraction(data):
    feature_ls = []
    for idx, img in enumerate(data):
        # Compute the Intergral image
        for i in range(img.ndim):
            img = img.cumsum(axis=i)
        # calculate kernels
        horizontal_kernel = [np.hstack((np.zeros((1, n)), np.ones((1, n)))) for n in range(1, int(img.shape[1]/2)-1)]
        vertical_kernel = [np.hstack((np.ones((n, 2)), np.zeros((n, 2)))) for n in range(1, int(img.shape[0]/2)-1)]
        # compute feature using kernels
        feature = []
        for kernel in horizontal_kernel:
            h, w = kernel.shape
            for j in range(img.shape[0] - 1):
                for i in range(img.shape[1] - w):
                    sum1 = sum_pixels(img, [j, i], [j, i + w/2], [j + 1, i], [j + 1, i + w/2])
                    sum2 = sum_pixels(img, [j, i + w/2], [j, i + w], [j + 1, i + w/2], [j + 1, i + w])
                    feature.append(sum2-sum1)
        for kernel in vertical_kernel:
            h, w = kernel.shape
            for j in range(img.shape[0] - h):
                for i in range(img.shape[1] - 2):
                    sum1 = sum_pixels(img, [j, i], [j, i + 2], [j + h/2, i], [j + h/2, i + 2])
                    sum2 = sum_pixels(img, [j + h/2, i], [j + h/2, i + 2], [j + h, i], [j + h, i + 2])
                    feature.append(sum1-sum2)
        feature = np.asarray(feature).flatten()
        feature_ls.append(feature)

    return np.asarray(feature_ls)


# train_pos_feature = Haar_feature_extraction(train_pos_data)
# train_neg_feature = Haar_feature_extraction(train_neg_data)
```

```python
train_data = np.concatenate((train_pos_data, train_neg_data), axis=0)          # (1420, 20, 40)
# print(train_data.shape)
train_label = np.concatenate((train_pos_label, train_neg_label), axis=1)      # (1, 1420)
# print(train_label.shape)
train_feature = Haar_feature_extraction(train_data)                          # (num_samples, num_features)
# print(train_feature.shape)

# ================================== training =====================================
def build_weak_classifier(feature, label):
    num_samples, num_feature = feature.shape
    num_pos, num_neg = np.sum(label), num_samples-np.sum(label)
    # Initialization
    wgt = np.concatenate((np.ones((num_pos, )) / (2 * num_pos), np.ones((num_neg, )) / (2 * num_neg)), axis=0)
    best_classifier_ls = []
    confidence_factor = []
    for t in range(20):
        # wgt normalization
        wgt = wgt / np.sum(wgt)
        sorted_wgt = [x for _, x in sorted(zip(feature, wgt))]
        sorted_label = [x for _, x in sorted(zip(feature, label))]
        # error estimation
        T_pos, T_neg = np.sum(wgt[:num_pos]), np.sum(wgt[num_pos:])
        S_pos, S_neg = np.cumsum(sorted_wgt * sorted_label), np.cumsum(sorted_wgt * np.abs(1 - sorted_label))
        err1, err2 = (S_pos + (T_neg - S_neg)), (S_neg + (T_pos - S_pos))
        min_err = np.minimum(err1, err2)
        min_err_idx = np.argmin(min_err)
        theta = feature[min_err_idx]
        polarity = ((err1[min_err_idx] <= err2[min_err_idx]) - 0.5) * 2
        if polarity == 1:
            classification = np.asarray((feature[t] >= theta) * 1, dtype=np.uint8)
        elif polarity == -1:
            classification = np.asarray((feature[t] < theta) * 1, dtype=np.uint8)
        else:
            print('Error detected')
        wrong_num_pred = np.sum(np.abs(classification - sorted_label))
        classifier_param = [feature[min_err_idx], polarity, wrong_num_pred]
        best_classifier_ls.append(classifier_param)
        # compute confidence factor and update weights
        beta_t = min_err / (1 - min_err)
        alpha_t = np.log(1 / beta_t)
        confidence_factor.append(alpha_t)
        wgt = wgt * (beta_t ** (np.abs(classification - sorted_label)))

    return best_classifier_ls, confidence_factor


best_weak_classifier, confidence_factor = build_weak_classifier(train_feature, train_label)
strong_classifier = np.asarray(np.matmul(best_weak_classifier, confidence_factor) > np.sum(0.5 * confidence_factor))


# ================================== testing =====================================
test_path = "ECE661_2020_hw11_DB2/test/"
test_pos_data, test_pos_label, test_neg_data, test_neg_label = load_img(test_path)
test_data = np.concatenate((test_pos_data, test_neg_data), axis=0)
train_label = np.concatenate((test_pos_label, test_neg_label), axis=1)
test_feature = Haar_feature_extraction(test_data)                            # (
```

```
        num_samples, num_features)
```

Task2.py