ECE 661 Homework 7

Weichen Xu

xu1363@purdue.edu

1. Gram-matrix Based Classification Using SVM

Step 1: Preprocessing the data

Since all images in the task do not necessarily have the same size, we need to convert them to a common size. We can either directly resize the image to a designated size, or we can crop images from the center to the size of image which has the minimal size in all images. In this case, my implementation is to resize all image to the shape of $(128,128)$. Besides for color images, we need to convert them into grayscale images.

Step 2: Build Gram-matrix based features

First, we need to build C different random 3×3 convolutional kernels, while the sum of kernel should be 0. Then convolve these convolutional kernels with the grayscale image, thus we can get C output convolution results, where we refer them as C channels. Then for each channel, we will downsample it to the size $K \times K$, here I choose $K = 16$. We can further flatten the 2D outputs into vectors of K^2 elements. For C vectors, we can take the inner products for all pairs to get the symmetric Gram matrix of size $C \times C$.

$$G_{ij} = \langle v_i, v_j \rangle$$

Since the Gram matrix is symmetric, we can represent the matrix using only its upper triangular matrix. An image can thus be represented with a point in $\frac{C(C+1)}{2}$-dimension space.

For $C = 3$, we have following Gram representation for an image,

$$G = \begin{bmatrix} a & b & c \\ b & d & e \\ c & e & f \end{bmatrix}$$

$$g = (a, b, c, d, e, f)$$

Step 3: Training with support vector machine (SVM)

For the training with SVM, we need to separate all training data into two subsets, training set and validation set. In my implementation, I randomly assign 70% to the training set, 30% to the validation set. Then use OpenCV library to generate the SVM model for training and obtain prediction accuracy for validation set.

Step 4: Run for N trials of training for best SVM model and convolutional kernels

For N trials of running step 1-3, we get the validation accuracy and SVM model and convolutional kernels at the end of each trial. Then save the SVM mode and convolutional kernels only when the validation accuracy improves comparing to current best validation accuracy.

Step 5: Test on testing images

Using saved SVM model and convolutional kernels to predict with the testing images, obtain the accuracy as the testing data accuracy.

2. Experiment Results

Trial #1
Training accuracy is 0.3034300791556728
Validation accuracy is 0.31384615384615383
svm has been saved

Trial #2
Training accuracy is 0.1741424802110818
Validation accuracy is 0.16615384615384615

Trial #3
Training accuracy is 0.27440633245382584
Validation accuracy is 0.27692307692307694

Trial #4
Training accuracy is 0.21108179419525067
Validation accuracy is 0.23692307692307693

Trial #5
Training accuracy is 0.3113456464379947
Validation accuracy is 0.3230769230769231
svm has been saved

Trial #6
Training accuracy is 0.23218997361477572
Validation accuracy is 0.24615384615384617

Trial #7
Training accuracy is 0.4179419525065963
Validation accuracy is 0.4569230769230769
svm has been saved

Trial #8
Training accuracy is 0.287598944591029
Validation accuracy is 0.23076923076923078
….

Number of trials	$N = 100$
Number of channels	$C = 20$
Number of features	$K = 16$
SVM kernel	Linear

Table 1: Parameter for best trained SVM model

		Prediction			
		Cloudy	Rain	Shine	Sunrise
Ground Truth	Cloudy	**6**	0	0	4
	Rain	1	**1**	2	6
	Shine	0	0	**2**	8
	Sunrise	0	0	0	**10**

Table 2: Confusion matrix for testing images

The overall accuracy is **47.5%**, it is not a very high result comparing to the deep learning methods, but it at least shows improvement than random guess accuracy **25%**.

From the confusion matrix we can notice that for all "sunrise" image, the trained SVM model give correct prediction. However, for testing images of other classes, the SVM model still tends to predict it as the "sunrise", it makes some sense that "shine" image which has very similar features as "sunrise" images, the SVM predict most such class's images as "sunrise". For other classes, the SVM model still largely likely to predict it as "sunrise", I suspect since the convolutional kernels are randomly generated, it cannot exactly represent the real features in the images.

3. Observations

For this classification task, I have tried with different set of C and different setting for SVM classifier, including kernel type, i.e. linear, polynomial, RBF kernels.

For C values, I start with $C = 4$, the training process is very fast. But for each trial, the validation accuracy never exceeds 30%, which naturally gives a poor testing accuracy around 20%, which is even worse than random guess. Then I go with higher C value, roughly speaking this does increase the overall accuracy. But when C is above 20, In my case I tried C as high as 100 it does not have enough positive affect on the accuracy.

For the selection of SVM model, I mainly experiment with two SVM kernel, linear and radial basis function (RBF). The RBF kernel tends to best fit the training data, which corresponds to the definition of such function, but returns same class prediction for validation and testing data. So I decide to go with the linear kernel, which is the easy and simple-minded method, and it gives reasonable predictions. The final SVM model still rely heavily on the convolutional kernels, however, in this case convolutional kernels are totally random.

4. Code

```
#!/usr/bin/env python
# coding: utf-8


import numpy as np
import cv2
import time
import os
import matplotlib.pyplot as plt
from scipy import signal
from sklearn import svm, metrics
from sklearn.metrics import confusion_matrix

def get_conv_mask(C):
    # create C of different convolutional masks
    masks = np.zeros((C,3,3))
    for i in range(C):
        conv_mask = np.random.uniform(-1, 1, size = (3,3))
        conv_mask = conv_mask - np.sum(conv_mask) / 9
        masks[i, :, :] = conv_mask
    return masks

def get_Gram_feature(img, masks):
    # masks contatin C different convolutional masks
    vec = []
    for i in range(masks.shape[0]):
        mask = masks[i,:,:]
        # obtain result of convolving image with mask
        img_conv = signal.convolve2d(img, mask, mode='same')
        img_output = cv2.resize(img_conv, (16, 16), interpolation = cv2.INTER_AREA)
        vec.append(img_output.flatten())

    vec = np.array(vec)
    gram = np.dot(vec, np.transpose(vec)) # obtain gram matrix

    # retain the upper triangular part of gram matrix
    vec_feature = []
    for i in range(C):
        for j in range(C):
            if j >= i:
                vec_feature.append(gram[i][j])

    return np.array(vec_feature)

def check_label(name):
    # obtain image label from file name
    if 'cloudy' in name:
        return 0
    if 'rain' in name:
        return 1
    if 'shine' in name:
        return 2
    if 'sunrise' in name:
```

```
        return 3

def seperate_dataset(inputs, labels):
    # seperate data into training data and validation data
    idx_arr = np.arange(len(inputs))
    np.random.shuffle(idx_arr)

    ratio = 0.7 # ratio of training data in all data
    idx_train, idx_test = idx_arr[: int(ratio * len(inputs))], idx_arr[int(ratio * len(inputs)) : ]

    inputs_train, labels_train = inputs[idx_train, :], labels[idx_train]
    inputs_test, labels_test = inputs[idx_test, :], labels[idx_test]


    return inputs_train, labels_train, inputs_test, labels_test

def training(N, C):
    training_path = '/home/xu1363/Documents/ECE 661/hw8/imagesDatabaseHW8/training/'
    # resizing image to common size
    h = 128
    w = 128
    accuracy_best = 0

    for n in range(N):
        print('Trial #' + str(n+1))
        labels = []
        inputs = []
        masks = get_conv_mask(C)
        for name in os.listdir(training_path):
            label = check_label(name)
            img = cv2.imread(training_path + name)
            img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            img = cv2.resize(img, (h, w), interpolation = cv2.INTER_AREA)
            vec_feature = get_Gram_feature(img, masks)

            labels.append(int(label))
            inputs.append(vec_feature)

        inputs = np.array(inputs)
        labels = np.array(labels)

        # build SVM model
        svm=cv2.ml.SVM_create()
        svm.setC(1)
        svm.setGamma(0.1)
        svm.setType(cv2.ml.SVM_C_SVC)
        svm.setKernel(cv2.ml.SVM_LINEAR)
        #svm.setKernel(cv2.ml.SVM_RBF)
        #svm.setTermCriteria((cv2.TERM_CRITERIA_MAX_ITER, 1000, 1e-6))

        # train
        inputs_train, labels_train, inputs_valid, labels_valid = seperate_dataset(inputs, labels)
        svm.train(np.float32(inputs_train), cv2.ml.ROW_SAMPLE, labels_train)

        _,preds =svm.predict(np.float32(np.array(inputs_train)))
```

```
        preds = preds.ravel()
        accuracy_train = metrics.accuracy_score(labels_train, preds)
        print('Training accuracy is ', accuracy_train)

        # validate
        _,preds =svm.predict(np.float32(np.array(inputs_valid)))
        preds = preds.ravel()
        accuracy_valid = metrics.accuracy_score(labels_valid, preds)
        print('Validation accuracy is ', accuracy_valid)

        # save model only when validation accuracy improves
        if accuracy_valid > accuracy_best:
            accuracy_best = accuracy_valid
            masks_best = masks
            svm.save('/home/xu1363/Documents/ECE 661/hw8/trained_svm.xml')
            print('svm has been saved')

        print('')

    # return with convolutional masks generating best validation accuracy
    return masks

def testing(masks):
    testing_path = '/home/xu1363/Documents/ECE 661/hw8/imagesDatabaseHW8/testing/'
    h = 128
    w = 128
    #C = 8
    labels = []
    inputs = []
    svm=cv2.ml.SVM_load('trained_svm.xml')
    for name in os.listdir(testing_path):
        label = check_label(name)
        img = cv2.imread(testing_path + name)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        img = cv2.resize(img, (h, w), interpolation = cv2.INTER_AREA)
        vec_feature = get_Gram_feature(img, masks)
        labels.append(int(label))
        inputs.append(vec_feature)

    _, preds=svm.predict(np.float32(np.array(inputs)))
    preds = preds.ravel()
    print(labels)
    print(preds)

    accuracy = metrics.accuracy_score(labels,preds)
    print('Test accuracy is', accuracy)
    print(metrics.confusion_matrix(labels, preds))


C = 20
N = 100
masks = training(N, C)
testing(masks)
```