Prateek Gupta

gupta198@purdue.edu

ECE 661 Homework 8

**Overview**

In this assignment we will use the gram matrix technique of characterizing the texture in each image and create a model to classify four types of weather patterns. We will use Support Vector Machine to classify each image. We will train the model with a set of approximately 1000 weather patterns and their true labels.

**Gram Matrix**

Computing the Gram Matrix is done by first choosing the number of channels. In the algorithm below, we will compute the Gram matrix for several channel numbers. So, we will let the number of channels be represented by C. We will start by down sampling the image to a 16x16 grayscale image. Then, that image will be convolved with C 3x3 matrices of random values between -1 and 1, such that the values add up to 0. Each convolution result is converted into a vector of one dimension. Then the inner product of each pair vectors is taken and formed into a C x C matrix. This forms the Gram matrix. Note that this matrix will be symmetric.

**Overall Algorithm**

We will start by separating a set of validation images from the training set. Then, on the training set, we will construct the feature space for each image based on the Gram Matrix. That is, we will flatten the upper triangular portion of the Gram Matrix (since it is symmetric, and we do not need to include the redundant information). Then, we will feed the feature space representation and the true labels into the Support Vector Machine (SVM) algorithm to generate a model.

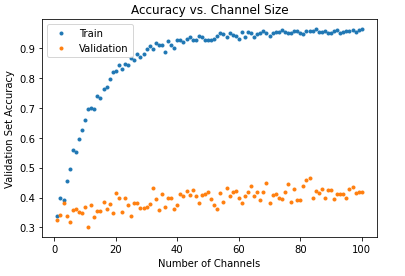
To check how accurate the model is, we will have it predict the labels of the validation set, by first finding the corresponding feature space representation of each image, and then those predicted labels against the true labels.

When setting up the Gram Matrix, we do have to select how many channels to use. So, we will run the above algorithm numerous times, each time updating the number of channels we use. We will store the model that gives us the best accuracy on the validation set.

Once we have the model that gives us the best accuracy on the validation set, we can run it on the testing set. We will use the same approach. That is, we will construct a gram matrix for each image in the set, using the number of channels that gave us the best model, and then use the best model to predict the labels of each image. Since we have the true labels, we will be able to construct a confusion matrix of the testing set.

**Results**

Running the algorithm for the number of channels being between 0 and 100, the best accuracy on the validation set was 46.67%, which corresponded to 83 channels. The plot below shows accuracy of each number of channels on both the training set, and the validation set.



Here, we observe that the accuracy on the training set tends to increase as the number of channels increases. This is likely because there is more opportunity for the model to overfit to the training data as the dimension of the feature space increases. However, this is why we use the model that gives us the best validation accuracy, because we want a model that will generalize to any set, not just to the training set.

Now that we have the best model with the optimal number of channels for the Gram matrix, we can introduce, for the first time, the given test set and predict its labels. The test set was be converted into feature vectors, and those matrices were be fed into the model. Since we did also have the true labels, the results are shown in the confusion matrix below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | True Label | | | |
| Cloudy | Rain | Shine | Sunrise |
| Predicted Label | Cloudy | 2 | 2 | 1 | 2 |
| Rain | 2 | 8 | 2 | 1 |
| Shine | 0 | 0 | 0 | 0 |
| Sunrise | 6 | 0 | 7 | 7 |

The accuracy is given by the sum of the correct prediction divided by the total. So on the testing set, we had an accuracy of 42.5%. While this is not great, it is certainly better than randomly guessing, which would give us an expected accuracy of 25%. This model did do a significantly better job than guessing randomly.

import numpy as np

import cv2

import random

from datetime import datetime

from sklearn.datasets.samples\_generator import make\_blobs

from sklearn.model\_selection import train\_test\_split

from matplotlib import pyplot as plt

from sklearn.svm import LinearSVC

from sklearn.metrics import confusion\_matrix

import pickle

from joblib import dump, load

dataset = []

for i in range (290):

im = cv2.imread("images/training/cloudy" + str(i+1) + ".jpg")

dataset.append([cv2.cvtColor(im,cv2.COLOR\_BGR2GRAY), "cloudy"])

for i in range (205):

im = cv2.imread("images/training/rain" + str(i+1) + ".jpg")

dataset.append([cv2.cvtColor(im,cv2.COLOR\_BGR2GRAY), "rain"])

for i in range (243):

im = cv2.imread("images/training/shine" + str(i+1) + ".jpg")

dataset.append([cv2.cvtColor(im,cv2.COLOR\_BGR2GRAY), "shine"])

for i in range (347):

im = cv2.imread("images/training/sunrise" + str(i+1) + ".jpg")

dataset.append([cv2.cvtColor(im,cv2.COLOR\_BGR2GRAY), "sunrise"])

random.shuffle(dataset)

train = dataset[:800]

validation = dataset[800:]

def getGram(im, C):

#downsample image

im = cv2.resize(im, (16,16))

vect = []

for i in range(C):

mat = np.random.rand(3,3) \* 2 - 1

mat = np.subtract(mat, np.sum(mat)/9)

conv = cv2.filter2D(im, -1, mat)

v = np.reshape(conv, (256, 1))

vect.append(v)

gram = []

for i in range(C):

for j in range(i, C):

gram.append(np.sum(vect[i] \* vect[j]))

return gram

epochs = 100 #change this to 100 later

maxAccuracy = 0

bestC = 1

accCTest = []

accCTrain = []

for e in range(epochs):

gramTrain = []

labelsTrain = []

for i in range(len(train)):

gramTrain.append(getGram(train[i][0], e+1))

labelsTrain.append(train[i][1])

gramTest = []

labelsTest = []

for i in range(len(validation)):

gramTest.append(getGram(validation[i][0], e + 1))

labelsTest.append(validation[i][1])

clf = svm.SVC()

clf.fit(gramTrain, labelsTrain)

accCTrain.append(clf.score(gramTrain, labelsTrain))

y\_pred = clf.predict(gramTest)

confusion = np.zeros((4, 4)) #cloudy, rain, shine, sunrise

for i in range(len(labelsTest)):

if labelsTest[i] == "cloudy":

if y\_pred[i] == "cloudy":

confusion[0][0] += 1

elif y\_pred[i] == "rain":

confusion[1][0] += 1

elif y\_pred[i] == "shine":

confusion[2][0] += 1

elif y\_pred[i] == "sunrise":

confusion[3][0] += 1

elif labelsTest[i] == "rain":

if y\_pred[i] == "cloudy":

confusion[0][1] += 1

elif y\_pred[i] == "rain":

confusion[1][1] += 1

elif y\_pred[i] == "shine":

confusion[2][1] += 1

elif y\_pred[i] == "sunrise":

confusion[3][1] += 1

elif labelsTest[i] == "shine":

if y\_pred[i] == "cloudy":

confusion[0][2] += 1

elif y\_pred[i] == "rain":

confusion[1][2] += 1

elif y\_pred[i] == "shine":

confusion[2][2] += 1

elif y\_pred[i] == "sunrise":

confusion[3][2] += 1

elif labelsTest[i] == "sunrise":

if y\_pred[i] == "cloudy":

confusion[0][3] += 1

elif y\_pred[i] == "rain":

confusion[1][3] += 1

elif y\_pred[i] == "shine":

confusion[2][3] += 1

elif y\_pred[i] == "sunrise":

confusion[3][3] += 1

accuracy = np.trace(confusion) / np.sum(confusion)

accCTest.append(accuracy)

if(accuracy > maxAccuracy):

print(confusion)

print(accuracy)

print(e+1)

maxAccuracy = accuracy

bestClf = clf

bestC = e+1

dump(clf, 'model.joblib')

plt.plot(range(1, len(accCTrain) + 1), accCTrain, '.')

plt.plot(range(1, len(accCTest) + 1), accCTest, '.')

plt.title('Accuracy vs. Channel Size')

plt.ylabel('Validation Set Accuracy')

plt.xlabel('Number of Channels')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()

clf = load('model.joblib')

testData = []

for i in range (290, 300):

im = cv2.imread("images/testing/cloudy" + str(i+1) + ".jpg")

testData.append([cv2.cvtColor(im,cv2.COLOR\_BGR2GRAY), "cloudy"])

for i in range (205, 215):

im = cv2.imread("images/testing/rain" + str(i+1) + ".jpg")

testData.append([cv2.cvtColor(im,cv2.COLOR\_BGR2GRAY), "rain"])

for i in range (243, 253):

im = cv2.imread("images/testing/shine" + str(i+1) + ".jpg")

testData.append([cv2.cvtColor(im,cv2.COLOR\_BGR2GRAY), "shine"])

for i in range (347, 357):

im = cv2.imread("images/testing/sunrise" + str(i+1) + ".jpg")

testData.append([cv2.cvtColor(im,cv2.COLOR\_BGR2GRAY), "sunrise"])

random.shuffle(testData)

gram = []

labels = []

for i in range(len(testData)):

gram.append(getGram(testData[i][0], bestC))

labels.append(testData[i][1])

y\_pred = bestClf.predict(gram)

confusion = np.zeros((4, 4)) #cloudy, rain, shine, sunrise

for i in range(len(labels)):

if labels[i] == "cloudy":

if y\_pred[i] == "cloudy":

confusion[0][0] += 1

elif y\_pred[i] == "rain":

confusion[1][0] += 1

elif y\_pred[i] == "shine":

confusion[2][0] += 1

elif y\_pred[i] == "sunrise":

confusion[3][0] += 1

elif labels[i] == "rain":

if y\_pred[i] == "cloudy":

confusion[0][1] += 1

elif y\_pred[i] == "rain":

confusion[1][1] += 1

elif y\_pred[i] == "shine":

confusion[2][1] += 1

elif y\_pred[i] == "sunrise":

confusion[3][1] += 1

elif labels[i] == "shine":

if y\_pred[i] == "cloudy":

confusion[0][2] += 1

elif y\_pred[i] == "rain":

confusion[1][2] += 1

elif y\_pred[i] == "shine":

confusion[2][2] += 1

elif y\_pred[i] == "sunrise":

confusion[3][2] += 1

elif labels[i] == "sunrise":

if y\_pred[i] == "cloudy":

confusion[0][3] += 1

elif y\_pred[i] == "rain":

confusion[1][3] += 1

elif y\_pred[i] == "shine":

confusion[2][3] += 1

elif y\_pred[i] == "sunrise":

confusion[3][3] += 1

accuracy = np.trace(confusion) / np.sum(confusion)

print(confusion)

print(accuracy)