Angel U. Ortega
Computer Vision – Lab 4
Version 1.0
11/26/2016

Table of Contents

1. Introduction

1.1. Lab Overview

The lab is composed of five problems, P1 – P6. The problem statement is as follows:

P1. Pixel clustering. Suppose you want to represent a color image as accurately as possible but have a limit on the number of colors you can display. If you have k colors available you could cluster the {r, g, b} values of the pixels in the image into k clusters, and then replace each pixel value by the mean of the cluster it is assigned to. Implement this and experiment with various values of k.

P2. Image clustering. Cluster the MNIST dataset using pixel intensities as features. Observe if images contained in the same cluster represent the same digits. More formally, partition the dataset into k clusters (experiment with k = 10 and k = 20) and determine the fraction of the instances in each cluster that belong to the majority class in that cluster.

P3. Image clustering. Cluster the CIFAR-10 dataset using color histograms as features. As in the previous question, observe if images contained in the same cluster represent objects of the same class.

P4. Image clustering. Cluster the CIFAR-10 dataset histograms of gradients as features. As in the previous question, observe if images contained in the same cluster represent objects of the same class.

P5. Image classification. Download the scikit machine learning toolbox and classify the MNIST dataset using the algorithm and feature set of your choice. Make sure you don't use the same examples for training and testing as this will give you overoptimistic results. Extra credit points will be given to the best results obtained in the class. For reference, state of the art error rates for this dataset are around 1%.

P6. Image classification. Classify the CIFAR-10 dataset using the algorithm and feature set of your choice. Make sure you don't use the same examples for training and testing. Extra credit points will be given to the best results obtained in the class. For reference, state of the art error rates for this dataset are around 5%.

1.2. References

[1] https://drive.google.com/open?id=0B_kWRxLZdmeJWERtV2ExMFhNdnc
[2] http://johnloeber.com/docs/kmeans.html
[3] http://stackoverflow.com/questions/31528800/how-to-implement-zca-whitening-python

2. Proposed Solution Design and Implementation

2.1. P1

For this problem, I investigated the best tool to conduct kmeans to retrieve the closest color average given k colors. I decided on using CV2. The program retrieves image as an array, and applies kmeans with the number of colors desired. The results are then used to modify a copy of the original image using only the k colors retrieved by kmeans. The new image is then saved in a specified directory.

2.2. P2

For this problem, I had to first extract the dataset MNIST. I utilized the code provided by Mario Landa during class for this purpose. I utilized the cv2.kmeans method like in P1 to cluster the images. As instructed by the problem statement I utilized k=10 and k=20 to compare results. The results can be seen in Section 3.2. To compare accuracy of my solution to another approach, I modified John Loeber's kmeans implementation [2] to cluster MNIST dataset. The results from both programs are compared in Section 3.2.

2.3. P3

For this problem, I had to first extract the dataset CIFAR-10. I utilized the code provided in class for this purpose. To be able to use color histograms as features, I had to implement a method (getCH) to get the color histogram of an image. Then I read in all images, retrieve the color histogram from each, and store the resultant in a different object. Next, I use kmeans from the previous question but using the color histograms as clustering features. Finally, I implemented two methods to visualize my results: getAccuracy and plotData. The getAccuracy method returns the calculated accuracy for each cluster. The plotData method gets the resultant from the getAccuracy method, and plots it. Results for P3 can be seen in Section 3.3.

2.4. P4

For this problem, I used same program to extract dataset as in P3. For P3, I utilized the implementation I had for HOG from Lab 2. I started by reading in the images from the dataset and retrieving the histogram of gradients from each. I saved the result in a different object and used kmeans to cluster the array containing the histograms of gradients. I used the getAccuracy and plotData methods from the previous problem to visualize accuracy. Results for P4 can be seen in Section 3.4.

2.5. P5

For this problem, I utilized the KNN classifier with k = 10 and k = 20. I used pixel intensities as features for the classification. I first had to load the training and testing sets. I used the code provided in class for this purpose. I used a minkowski metric of 2 for the Euclidean standard. I fit the classifier with training set, then retrieve the score of both training set and the testing set. Results for P5can be seen in Section 3.5

2.6. P6

For this problem, I used a multi-layer perceptron classifier. I used pixel intensities and histograms of gradients as features. I used batch 1 as training and batch 3 as testing from the CIFAR-10 dataset. For this problem, I used both grayscale and color images. I utilized ZCA whitening normalization for pixel intensity features. For ZCA, I followed Timshel's implementation [3]. For HOG, I utilized the same implementation from P4. I applied whitening normalization and histogram of gradients to bot training and testing sets; I fit the resultants into two MLP classifiers. Results for P6 can be seen in Section 3.6.

3. Experimental Results and Conclusions

All images used for this lab can be accessed at:
https://drive.google.com/open?id=0B_kWRxLZdmeJWERtV2ExMFhNdnc [1]

3.1. P1

Figures 1A and 1B show results for P1. As expected, the greater the number of colors retrieved from original image, the more accurate the pixel cluster image is to the original. For example, in Figure 1A, the original image is a picture of Mount Fuji. With k = 1, the result is a single-color image. With k = 2, we can start to see the separation between sky and ground, but Mount Fuji is completely lost in the background. At k = 5, we can clearly see Mount Fuji, the sky, separation between horizon and foreground, and details in the flowers. There is also a color variation between the area that encased the flowers in k=2, and the same area in k=5. In k = 10 we can see a lot more detail on the flowers as well as the buildings and a greater variation in the sky colors. Finally, in k = 25, the image is very detailed, with at least 5 different shades of blue making up the sky. The same behavior can be seen in Figure 1B. The k = 1 image is a single-color image. With k = 2, the image is well defined, with a lot of contrast. As k increases, the details become more prominent in the image.

Figure 1A: From top to bottom and left to right - Original Image, and Pixel Clustering Images (k = 1, 2, 5, 10, 25)

Figure 1B: From top to bottom and left to right - Original Image, and Pixel Clustering Images (k = 1, 2, 5, 10, 25)

3.2. P2

Figures 2 and 3 show the results from my implementation of k-means and Loeber's modified implementation from pixel intensity image clustering of MNIST using k = 10 and k = 20. In comparison, my implementation receives almost identical results. Figure 3 shows the resulting images using k = 10. Certain resulting images seem very accurate, like "0" and "8". In comparison, however, others seem very blurry and like a mixture of multiple digits "3", "4", and "5". Finally, some digits are repeated, like "9", which might be a result of the program clustering similar digits together like "7" and "9".

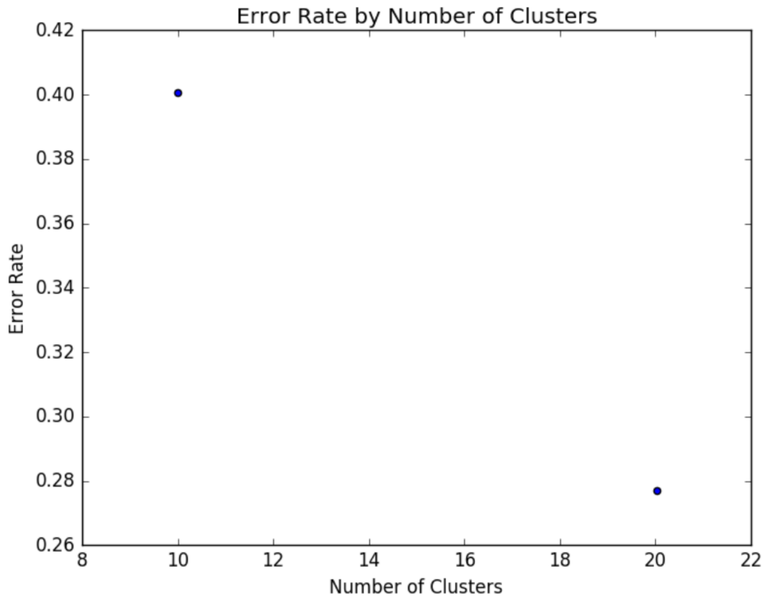

Figure 2: P2 implementation results

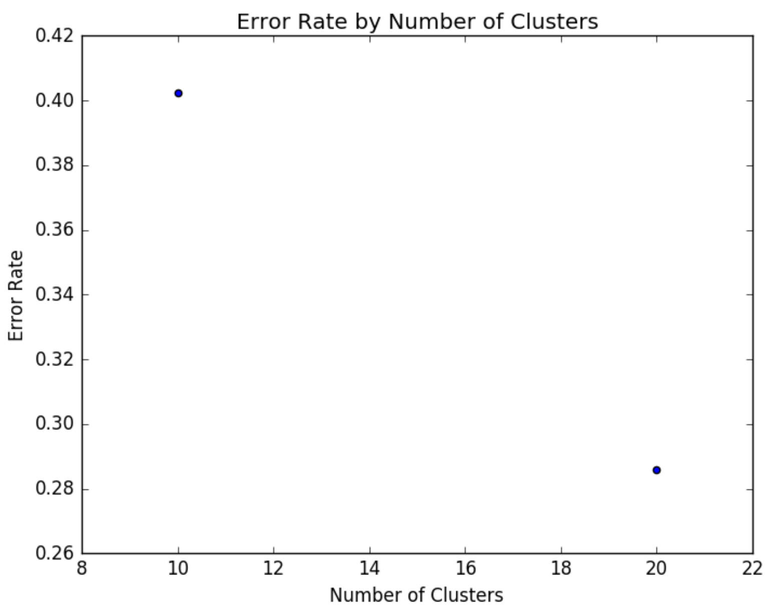

Figure 3: Loeber's Implementation Results

Figure 3: Image results for P2 using k = 10.

Figure 4: Image results for P2 using k = 20.

3.3. P3

Figures 5 and 6 shows the accuracy results for P3. From the results, I would conclude that color histograms do not make for a good feature to cluster CIFAR-10 images. Apart from the single almost perfect accuracy of one cluster, the other clusters receive very low results. Proportionally, using k = 10 gave better results than using k = 20.

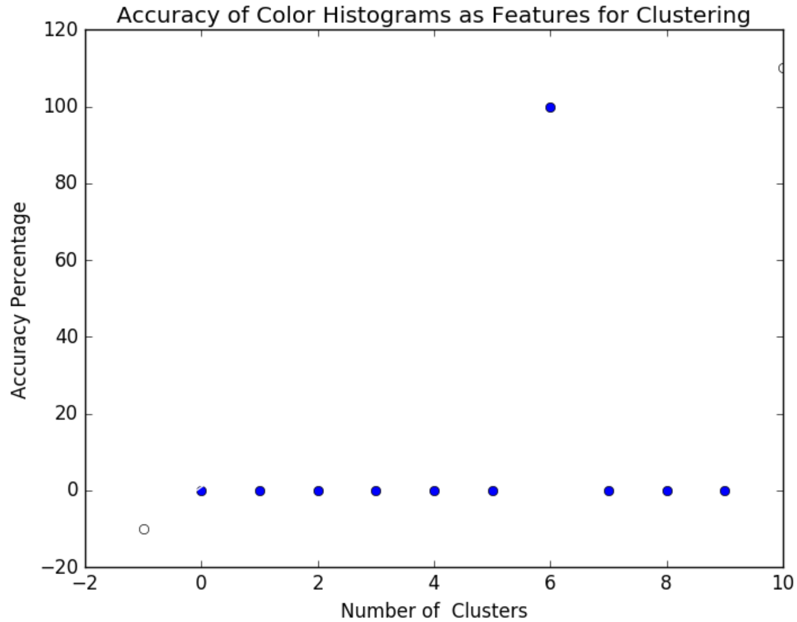

Figure 5: Accuracy of color histograms as features for clustering using k = 10

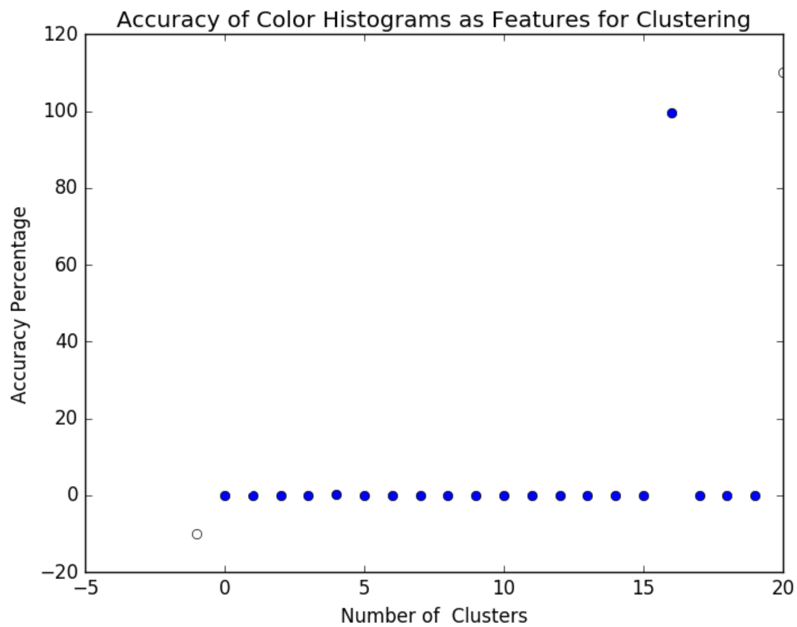

Figure 6: Accuracy of color histograms as features for clustering using k = 20

3.4. P4

Figure :

Figures 7 and 8 show the accuracy results of using HOG as features for image clustering CIFAR-10. In comparison to color histograms as features for clustering, HOG had better accuracy results. While certain clusters were still very low with near-zero accuracy, half of the clusters had accuracy above 20% for k = 10. Like in P3, using k = 10 gave better results, on average.

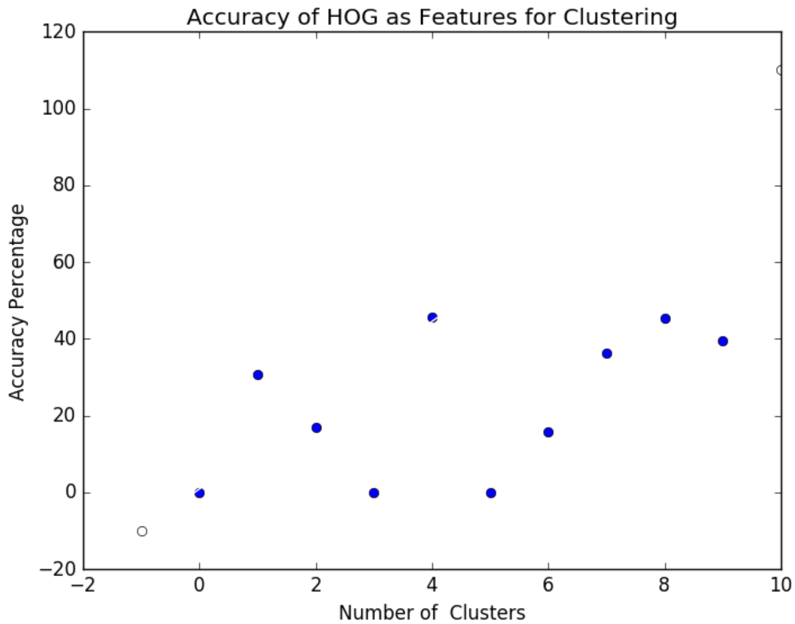

Figure 7: Accuracy of HOG as features for clustering using k = 10

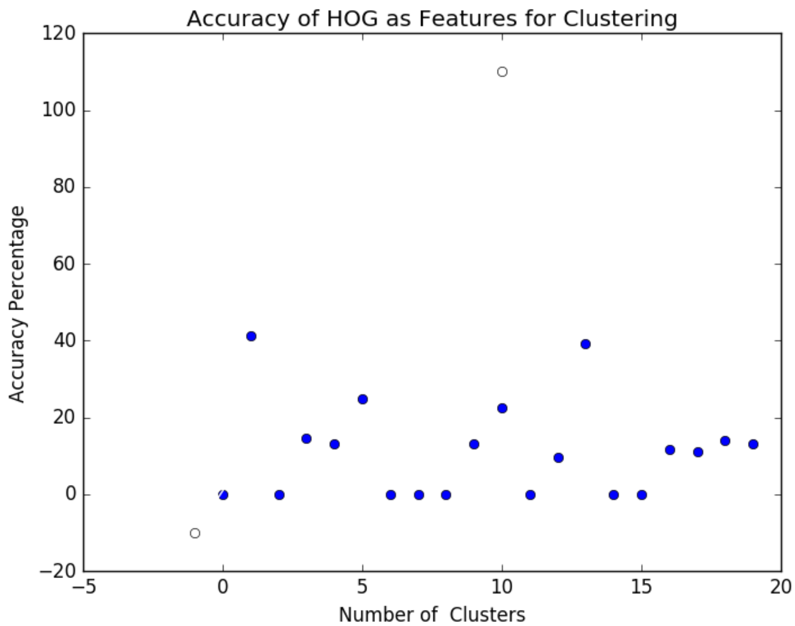

Figure 8: Accuracy of HOG as features for clustering using k = 20

3.5. P5

Figures 9 and 10 show the results for P5. I used k = 10 and k = 20 for the nearest neighbor classification. The blue dot in each graph represents the training set accuracy. The red dot represents the testing set accuracy. As expected, in both, the training had higher accuracy than the testing. A smaller number of neighbors returned a better accuracy.

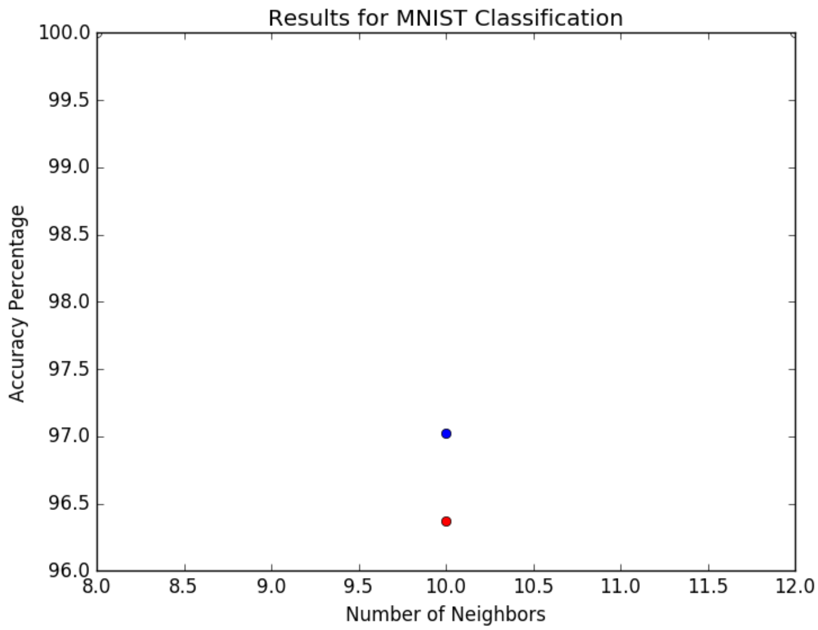

Figure 9: Results for P5 with 10 neighbors

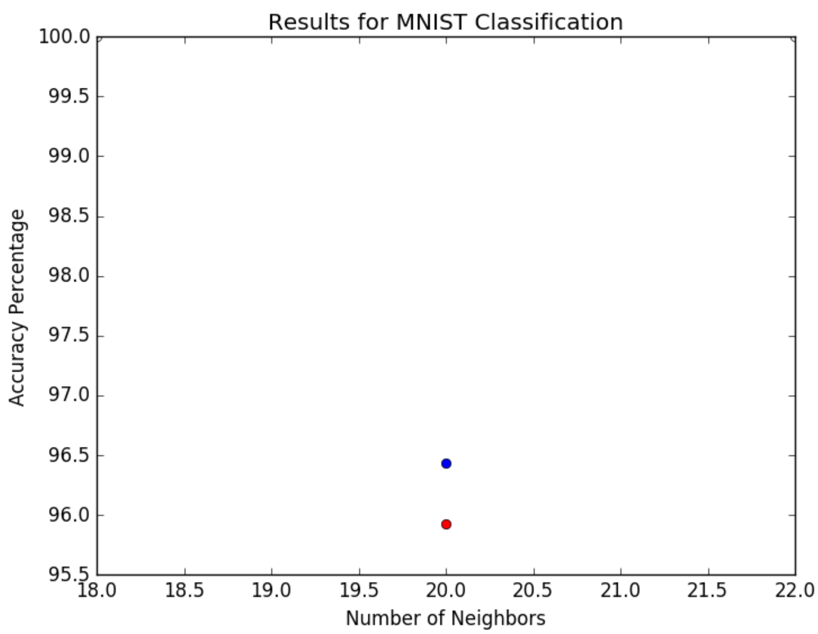

Figure 10: Results for P5 with 20 neighbors

3.6. P6

Accuracy of classification for CIFAR-10 was much lower than for MNIST. Pixel intensity as features had a much lower accuracy than HOG. Pixel intensity returned accuracy of approximately 11%. In comparison, using HOG returned accuracy of about 25%.

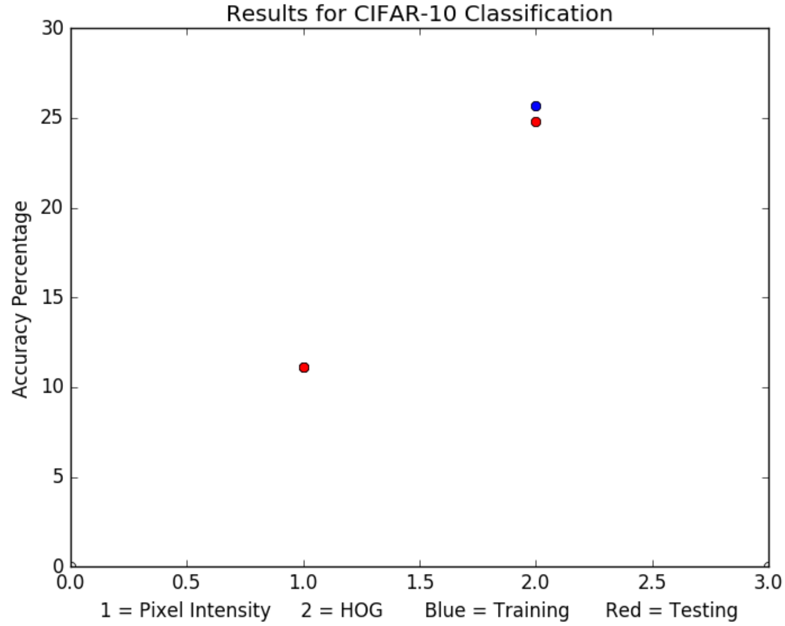

Figure 11: Results for P6

4. Appendix

4.1. P1 Code

```
# -*- coding: utf-8 -*-
"""
Created on Wed Oct 19 16:07:54 2016

@author: auort

Use k means clustering to posterize an image with n number of colors

"""

import numpy as np
import cv2

def getarray(img):
    array = img.reshape((-1,3))
    array = np.float32(array) #cv2.kmeans needs float array, soarray must be
converted
    return array

def getimage(array, label, img):
    array = np.uint8(array)#convert from float to int
    result = array[label.flatten()]
```

```
    resultimage = result.reshape((img.shape))#convert to original shape
    return resultimage

def posterize(imagename, numcolors, savedirectory):
    img = cv2.imread(imagename)
    array = getarray(img)
    criteria   =   (cv2.TERM_CRITERIA_EPS   +   cv2.TERM_CRITERIA_MAX_ITER,   10,
1.0)#stop iteration if epsilon is reached or if max iterations has occured; max
iterations = 10, epsilon = 1.0
    ret,label,result=cv2.kmeans(array,    numcolors,    None,    criteria,    10,
cv2.KMEANS_RANDOM_CENTERS)
    resultimage = getimage(result, label, img)
    cv2.imwrite(savedirectory, resultimage)

    '''
    cv2.imshow('posterized image using k means with k=' + str(numcolors) +' of
'+ imagename, resultimage)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    '''
posterize('fujismall.jpg', 1, "fuji1.jpg")
'''
posterize('fujismall.jpg', 2, 'fuji2.jpg')
posterize('fujismall.jpg', 5, 'fuji5.jpg')
posterize('fujismall.jpg', 10, 'fuji10.jpg')
posterize('fujismall.jpg', 25, 'fuji25.jpg')
'''
```

4.2. P2 Code

```
# -*- coding: utf-8 -*-
"""
Created on Tue Nov 8 09:44:32 2016

@author: auort
"""

from pylab import *
import numpy as np
import cv2
from loadMNIST import load_mnist
import matplotlib.pyplot as plt

def classify_digit(digit, labelled_centroids):
    print "classify_digit called!"

    mindistance = float("inf")
    for (label, centroid) in labelled_centroids:
        distance = np.linalg.norm(centroid - digit)
        if distance < mindistance:
            mindistance = distance
            closest_centroid_label = label
    return closest_centroid_label

def get_error_rate(digits,labelled_centroids):
    print "get_error_rate called!"
```

```
    classified_incorrect = 0
    for (label,digit) in digits:
        classified_label = classify_digit(digit, labelled_centroids)
        if classified_label != label:
            classified_incorrect +=1
    error_rate = classified_incorrect / float(len(digits))
    return error_rate

trainImages, trainLabels = load_mnist('training', digits=[0,1,2,3,4,5,6,7,8,9])
trainImages = np.float32(trainImages)
trainLabels = trainLabels.flatten()

# Define criteria = ( type, max_iter = 10 , epsilon = 1.0 )
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)

# Set flags (Just to avoid line break in the code)
flags = cv2.KMEANS_RANDOM_CENTERS

error_rates = {x:None for x in range(10,,10)+[100]}

k = 10

# Apply KMeans
compactness,labels,centers = cv2.kmeans(trainImages,k,None,criteria,10,flags)

error_rate = get_error_rate(centers, labels)
error_rates[k] = error_rate

k = 20

# Apply KMeans
compactness,labels,centers = cv2.kmeans(trainImages,k,None,criteria,10,flags)

error_rate = get_error_rate(centers, labels)
error_rates[k] = error_rate

# Show the error rates
x_axis = sorted(error_rates.keys())
y_axis = [error_rates[key] for key in x_axis]
plt.figure()
plt.title("Error Rate by Number of Clusters")
plt.scatter(x_axis, y_axis)
plt.xlabel("Number of Clusters")
plt.ylabel("Error Rate")
plt.show()

'''
i = 0
for i in range(k):
    img1 = (centers[i,:])
    img1 = np.reshape(img1, (28,28))
    cv2.imwrite("centroidsB20_%d" % (i,) + ".png", img1)
    figure()
    gray()
    imshow(img1)
'''
```

```
# -*- coding: utf-8 -*-
"""
Created on Thu Nov 17 14:20:08 2016

@author: auort
"""

import random
from base64 import b64decode
from json import loads
import numpy as np
import matplotlib.pyplot as plt
import time


def parse(x):
    print "parse called!"
    """
    to parse the digits file into tuples of
    (labelled digit, numpy array of vector representation of digit)
    """
    digit = loads(x)
    array = np.fromstring(b64decode(digit["data"]),dtype=np.ubyte)
    array = array.astype(np.float64)
    return (digit["label"], array)

with open("digits.base64.json","r") as f:
    digits = map(parse, f.readlines())

# pick a ratio for splitting the digits list into a training and a validation
set.
ratio = int(len(digits)*0.25)
validation = digits[:ratio]
training = digits[ratio:]

def display_digit(digit, labeled = True, title = ""):
    print "display_digit called!"
    """
    graphically displays a 784x1 vector, representing a digit
    """
    if labeled:
        digit = digit[1]
    image = digit
    plt.figure()
    fig = plt.imshow(image.reshape(28,28))
    fig.set_cmap('gray_r')
    fig.axes.get_xaxis().set_visible(False)
    fig.axes.get_yaxis().set_visible(False)
    if title != "":
        plt.title("Inferred label: " + str(title))

# writing Lloyd's Algorithm for K-Means clustering.
# (This exists in various libraries, but it's good practice to write by hand.)
def init_centroids(labelled_data,k):
    print "init_centroids called!"
```

```
    """
    randomly pick some k centers from the data as starting values for centroids.
    Remove labels.
    """
    return map(lambda x: x[1], random.sample(labelled_data,k))

def sum_cluster(labelled_cluster):
    print "sum_cluster called!"
    sum_ = labelled_cluster[0][1].copy()
    for (label,vector) in labelled_cluster[1:]:
        sum_ += vector
    return sum_

def mean_cluster(labelled_cluster):
    print "mean_cluster called!"
    sum_of_points = sum_cluster(labelled_cluster)
    mean_of_points = sum_of_points * (1.0 / len(labelled_cluster))
    return mean_of_points

def form_clusters(labelled_data, unlabelled_centroids):
    print "form_clusters called!"
    # enumerate because centroids are arrays which are unhashable,
    centroids_indices = range(len(unlabelled_centroids))

    # initialize an empty list for each centroid. The list will contain
    # all the datapoints that are closer to that centroid than to any other.
    # That list is the cluster of that centroid.
    clusters = {c: [] for c in centroids_indices}

    for (label,Xi) in labelled_data:
        # for each datapoint, pick the closest centroid.
        smallest_distance = float("inf")
        for cj_index in centroids_indices:
            cj = unlabelled_centroids[cj_index]
            distance = np.linalg.norm(Xi - cj)
            if distance < smallest_distance:
                closest_centroid_index = cj_index
                smallest_distance = distance
        # allocate that datapoint to the cluster of that centroid.
        clusters[closest_centroid_index].append((label,Xi))
    return clusters.values()

def move_centroids(labelled_clusters):
    print "move_centroids called!"
    """
    returns a list of centroids corresponding to the clusters.
    """
    new_centroids = []
    for cluster in labelled_clusters:
        new_centroids.append(mean_cluster(cluster))
    return new_centroids

def          repeat_until_convergence(labelled_data,          labelled_clusters,
unlabelled_centroids):
    print "repeat_until_convergence called!"
    previous_max_difference = 0
    while True:
```

```
        unlabelled_old_centroids = unlabelled_centroids
        unlabelled_centroids = move_centroids(labelled_clusters)
        labelled_clusters = form_clusters(labelled_data, unlabelled_centroids)
        differences      =      map(lambda      a,      b:      np.linalg.norm(a-
b),unlabelled_old_centroids,unlabelled_centroids)
        max_difference = max(differences)
        difference_change                 =                 abs((max_difference-
previous_max_difference)/np.mean([previous_max_difference,max_difference]))     *
100
        previous_max_difference = max_difference

        if np.isnan(difference_change):
            break
    return labelled_clusters, unlabelled_centroids

def cluster(labelled_data, k):
    print "cluster called!"
    """
    runs  k-means  clustering  on  the  data.  It  is  assumed  that  the  data  is
labelled.
    """
    centroids = init_centroids(labelled_data, k)
    clusters = form_clusters(labelled_data, centroids)
    final_clusters,  final_centroids  =  repeat_until_convergence(labelled_data,
clusters, centroids)
    return final_clusters, final_centroids

def assign_labels_to_centroids(clusters, centroids):
    print "assign_labels_to_centroids called!"

    labelled_centroids = []
    for i in range(len(clusters)):
        labels = map(lambda x: x[0], clusters[i])
        # pick the most common label
        most_common = max(set(labels), key=labels.count)
        centroid = (most_common, centroids[i])
        labelled_centroids.append(centroid)
    return labelled_centroids

def classify_digit(digit, labelled_centroids):
    print "classify_digit called!"

    mindistance = float("inf")
    for (label, centroid) in labelled_centroids:
        distance = np.linalg.norm(centroid - digit)
        if distance < mindistance:
            mindistance = distance
            closest_centroid_label = label
    return closest_centroid_label

def get_error_rate(digits,labelled_centroids):
    print "get_error_rate called!"

    classified_incorrect = 0
    for (label,digit) in digits:
        classified_label = classify_digit(digit, labelled_centroids)
        if classified_label != label:
```

```
            classified_incorrect +=1
    error_rate = classified_incorrect / float(len(digits))
    return error_rate

error_rates = {x:None for x in range(5,10,10)+[100]}

for k in range(5,10,10):
    print "K: %s" % k
    start_time = time.time()
    trained_clusters, trained_centroids = cluster(training, k)
    labelled_centroids       =       assign_labels_to_centroids(trained_clusters, 
trained_centroids)
    error_rate = get_error_rate(validation, labelled_centroids)
    error_rates[k] = error_rate
    end_time = time.time()
    run_time = end_time - start_time
    print "Runtime: %s seconds" % run_time

print "error_rates:"
print error_rates

# Show the error rates
x_axis = sorted(error_rates.keys())
y_axis = [error_rates[key] for key in x_axis]
plt.figure()
plt.title("Error Rate by Number of Clusters")
plt.scatter(x_axis, y_axis)
plt.xlabel("Number of Clusters")
plt.ylabel("Error Rate")
plt.show()
```

4.3. P3 Code

```
# -*- coding: utf-8 -*-
"""
Created on Thu Nov 17 14:15:54 2016

@author: auort
"""
import matplotlib.pyplot as plt
import numpy as np

def getAccuracy(derivLabels,k,origLabels):
    print(origLabels)
    data = np.zeros(k)
    for i in range(0,k):
        acc = [1 if derivLabels[x] == i else 0 for x in derivLabels]
        print(derivLabels)
        acc = np.multiply(acc,origLabels)
        print(acc)
        acc = [x for x in acc if x>0]
        data[i]=[i,acc]
    return data

def plotData(data, boundaries, gtitle, xlbl, ylbl):
    plt.plot(*zip(*data), marker='o', color='b', ls='')
    plt.plot(*zip(*boundaries), marker='o', color='w')
```

```
    plt.title(gtitle)
    plt.xlabel(xlbl)
    plt.ylabel(ylbl)
    plt.show()



# -*- coding: utf-8 -*-
"""
Created on Thu Nov 17 15:29:11 2016

@author: auort
"""

from loadCIFAR2 import load_cifar
import numpy as np
import cv2
import plot

def getCH(Im,n):
    colorHistogram = np.zeros((3,n))
    barSize = 255/(n-1)
    for i in range(len(Im)):
        for j in range(len(Im[i])):
            for k in range(3):
                index = Im[i][j][k]/barSize
                colorHistogram[k][index] = colorHistogram[k][index]+1
    return colorHistogram.flatten()

images,         trainLabels         =         load_cifar("color","data_batch_1",
"C:/Users/auort/Desktop/CV_L4/RCode/cifar-10-batches-py/")
bins = 16
dataCH = np.zeros((len(trainLabels),bins*3))
for i in trainLabels:
    dataCH[i]= getCH(images[i], bins)

dataCH = np.float32(dataCH)
# Define criteria = ( type, max_iter = 10 , epsilon = 1.0 )
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
# Set flags (Just to avoid line break in the code)
flags = cv2.KMEANS_RANDOM_CENTERS

k = 20

# Apply KMeans
compactness,labels,centers = cv2.kmeans(dataCH,k,None,criteria,10,flags)
labels = labels.flatten()

data = plot.getAccuracy(labels,k,trainLabels)
title = "Accuracy of Color Histograms as Features for Clustering"
xlabel = "Number of  Clusters"
ylabel = "Accuracy Percentage"
plot.plotData(data,[[-1,0],[k+1,100]], title, xlabel, ylabel)
```

4.4. P4 Code

```
"""
Created on Sat Nov 19 18:06:43 2016

@author: auort
"""
from PIL import Image
from collections import *
from loadCIFAR2 import load_cifar
import plot
from HOG import imHOG
import numpy as np
import cv2


images,       trainLabels       =       load_cifar("blackWhite","data_batch_1",
"C:/Users/auort/Desktop/CV_L4/RCode/cifar-10-batches-py/")
div,bins = 4,4
histData = np.zeros((len(trainLabels),div*div,bins))
for i in range(len(images)):
    histData[i]= imHOG(images[i], div, bins)

histData = np.float32(histData)
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
flags = cv2.KMEANS_RANDOM_CENTERS

k = 10
compactness,labels,centers = cv2.kmeans(histData,k,None,criteria,10,flags)
labels = labels.flatten()

data = plot.getAccuracy(labels,k,trainLabels)
title = "Accuracy of Color Histograms as Features for Clustering"
xlabel = "Number of Clusters"
ylabel = "Accuracy Percentage"
plot.plotData(data,[[-1,0],[k+1,100]], title, xlabel, ylabel)
```

4.5. P5 Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat Nov 19 14:48:28 2016

@author: auort
"""

import numpy as np
from loadMNIST import load_mnist
from sklearn.neighbors import KNeighborsClassifier
#from collections import *
import matplotlib.pyplot as plt

def train():
    trainIm, trainLb = load_mnist('training', digits=[0,1,2,3,4,5,6,7,8,9])
    trainIm = np.float32(trainIm).reshape((len(trainIm), -1))
    trainLb = trainLb.flatten()
    return trainIm, trainLb
```

```
def test():
    testIm, testLb = load_mnist('testing', digits=[0,1,2,3,4,5,6,7,8,9])
    testIm = np.float32(testIm).reshape((len(testIm), -1))
    testLb = testLb.flatten()
    return testIm, testLb

def plotResult(training, testing, boundaries, gtitle, xlbl, ylbl):
    plt.plot(*zip(*training), marker='o', color='b', ls='')
    plt.plot(*zip(*testing), marker='o', color='r', ls='')
    plt.plot(*zip(*boundaries), marker='o', color='w')
    plt.title(gtitle)
    plt.xlabel(xlbl)
    plt.ylabel(ylbl)
    plt.show()

def classify(k):
    trainImages, trainLabels = train()
    testImages, testLabels = test()
    classifier = KNeighborsClassifier(n_neighbors=k,p=2)
    classifier.fit(trainImages[:30000], trainLabels[:30000])

    training = [[1,classifier.score(trainImages, trainLabels)]]
    testing = [[1,classifier.score(testImages, testLabels)]]
    boundaries = [[k-2,100],[k+2,100]]
    title = "Accuracy of Classification"
    xlabel = "Number of Neighbors
    ylabel = "Accuracy Percentage"
    plotResult(training, testing, boundaries, title, xlabel, ylabel)

classify(10)
classify(20)
```

4.6. P6 Code

```
"""
Created on Mon Nov 21 09:53:28 2016

@author: auort
"""
import numpy as np
from loadCIFAR2 import load_cifar
from sklearn.neural_network import MLPClassifier
from HOG import imHOG
import matplotlib.pyplot as plt
import plot

def plotResult(training, testing, boundaries, gtitle, xlbl, ylbl):
    plt.plot(*zip(*training), marker='o', color='b', ls='')
    plt.plot(*zip(*testing), marker='o', color='r', ls='')
    plt.plot(*zip(*boundaries), marker='o', color='w')
    plt.title(gtitle)
    plt.xlabel(xlbl)
    plt.ylabel(ylbl)
    plt.show()
```

```
def zca_whitening_matrix(X):
    """
    Function to compute ZCA whitening matrix (aka Mahalanobis whitening).
    INPUT:  X: [M x N] matrix.
        Rows: Variables
        Columns: Observations
    OUTPUT: ZCAMatrix: [M x M] matrix
    """
    sigma = np.cov(X, rowvar=True)
    U,S,V = np.linalg.svd(sigma)
    epsilon = 1e-5
    ZCAMatrix = np.dot(U, np.dot(np.diag(1.0/np.sqrt(S + epsilon)), U.T)) # [M x
M]
    return ZCAMatrix


trainImages, trainLabels = load_cifar("color","data_batch_1", "
C:/Users/auort/Desktop/CV_L4/RCode/cifar-10-batches-py/")
trainImages = np.float32(trainImages).reshape((len(trainImages), -1))
zxtrain = zca_whitening_matrix(trainImages)
trainImagesX = np.dot(zxtrain, trainImages)
trainLabels = np.array(trainLabels).flatten()

testImages, testLabels = load_cifar("color","data_batch_3", "
C:/Users/auort/Desktop/CV_L4/RCode/cifar-10-batches-py/")
testImages = np.float32(testImages).reshape((len(testImages), -1))
zxtest = zca_whitening_matrix(testImages)
testImagesX = np.dot(zxtest, testImages)
testLabels = np.array(testLabels).flatten()

classNorm = MLPClassifier()
classNorm.fit(trainImagesX, trainLabels)
#predictedLabels = classNorm.predict(testImages)

scoreNormTrain = classNorm.score(trainImagesX, trainLabels)
scoreNormTest = classNorm.score(testImagesX, testLabels)

hogImages, hogLabels = load_cifar("blackWhite","data_batch_1", "
C:/Users/auort/Desktop/CV_L4/RCode/cifar-10-batches-py/")
hogTestImages, hogTestLabels = load_cifar("blackWhite","data_batch_3", "
C:/Users/auort/Desktop/CV_L4/RCode/cifar-10-batches-py/")
div,bins = 4,4
histData = np.zeros((len(trainLabels),div*div*bins))
histTestData = np.zeros((len(trainLabels),div*div*bins))

for i in range(len(hogImages)):
    histData[i]= imHOG(hogImages[i], div, bins).flatten()
    histTestData[i]= imHOG(hogTestImages[i], div, bins).flatten()
```

```
classHOG = MLPClassifier()
classHOG.fit(histData, hogLabels)

scoreHOGTrain = classHOG.score(histData, hogLabels)
scoreHOGTest = classHOG.score(histTestData, hogTestLabels)

training = [[1, scoreNormTrain],[2, scoreHOGTrain]]
testing = [[1, scoreNormTest],[2, scoreHOGTest]]
boundaries = [[0,0],[3,0]]
title = "Results for CIFER-10 Classification"
xlabel = "1 = Pixel Intensity      2 = HOG        Blue = Training       Red = Testing"
ylabel = "Accuracy Percentage"
plotResult(training, testing, boundaries, title, xlabel, ylabel)
```

End of Document

End of Document