# CS416 – HW4

## 1 SVD (20 points)

We can use Eigenface to recognize faces. In this assignment you will represent a face image through Eigenfaces. You will be using face images from the Yale Face Database B where there are 10 faces under 64 lighting conditions.

You can download the data for this assignment at

http://www.cs.ucsd.edu/classes/sp05/cse152/faces.zip

For more information on the Yale Face Database, see

http://cvc.cs.yale.edu/cvc/projects/yalefacesB/yalefacesB.html

Take each 50x50 pixel training image and vectorize into an 2500-dimensional vector. Stack each image (a 2500-dimensional vector) as a row vector in a matrix $X$, the entire set of training image vectors. First zero center the data by subtracting the average image $mu$ from all images and get $X$, the entire set of training image vectors. Then perform Singular Value Decomposition (SVD) on the entire set of training image vectors, retaining the first k principal components.

$$U, \Sigma, V^T = SVD(X)$$

Choose k such that

$$\frac{\sum_{i=1}^{k} \sigma_i^2}{\sum_{i=1}^{r} \sigma_i^2} > 90\%$$

where $r$ is the dimension of $\Sigma$, i.e., the number of singular values. The k principal components (the first k row vectors of $V^T$), when converted back to images, are the Eigenfaces, and the span of the k principal components is the Eigenspace. Display the k eigenfaces as images in your program.

Let $V'^T$ be the matrix of the first k row vectors of $V^T$. That is, $V'^T$ is a truncated matrix of $V^T$.

Now pick an image from your data set and convert it to a column vector $x$ as described before. First center it by subtracting your previously calculated $mu$,

$x' = x - mu$. $w = V'^T * x'$ gives the projection of $x'$ to Eigenspace. That is, we have projected a high dimension data point (e.g., an image) to a low dimension data point. Use $mu$, $w$, and $V'^T$ to recover the image: First use $w$ to get $y' = V' * w$, then add back $mu$ by running $y = y' + mu$, which is your reconstructed image.

Display the original image and the new image and compare them. The two images should be similar, which means we can use the low dimension data to recover the high dimension data.

The following is an example that shows some useful functions.

```
import numpy as np
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

        img_name="pic1.jpg"
        # Read image
        img=mpimg.imread(img_name)
        # Reshape image
        fimg = np. reshape(img, 50*50)

        # SVD
        u, s, vh = np.linalg.svd(X, full_matrices=False)

        # Show two images
        f = plt.figure()
        f.add_subplot(1,2, 1)
        imgplot = plt.imshow(img1)
        f.add_subplot(1,2, 2)
        imgplot = plt.imshow(img2)
```

Save your program as eigenface.ipynb

# 2 Decision Tree (20 points)

Consider the following dataset consisting of five training examples followed by three test examples:

| x1 | x2 | x3 | y |
|---|---|---|---|
| - | + | + | - |
| + | + | + | + |
| - | + | - | + |
| - | - | + | - |
| + | + | - | + |
| + | - | - | ? |
| - | - | - | ? |
| + | - | + | ? |

There are three attributes (or features or dimensions), x1, x2 and x3, taking the values + and -. The label (or class) is given in the last column denoted y; it also takes the two values + and -.

Simulate each of the following learning algorithms on this dataset. In each case, show the final hypothesis that is induced, and show how it was computed. Also, say what its prediction would be on the three test examples.

- The decision tree algorithm discussed in class. For this algorithm, use the information gain (entropy) impurity measure as a criterion for choosing an attribute to split on. Grow your tree until all nodes are pure, but do not attempt to prune the tree.
- AdaBoost. For this algorithm, you should interpret label values of + and - as the real numbers +1 and -1. Use decision stumps as weak hypotheses, and assume that the weak learner always computes the decision stump with minimum error on the training set weighted in AdaBoost algorithm. Note that a decision stump is a one-level decision tree. Run your boosting algorithm for three rounds and list the intermediate results.

# 3 Naive Bayes (10 points)

The following table contains training examples that help predict whether a person is likely to have come kind of disease.

| ID | PAIN? | MALE? | SMOKES? | WORK OUT? | DISEASE? |
|---|---|---|---|---|---|
| 1. | yes | yes | no | yes | yes |
| 2. | yes | yes | yes | no | yes |
| 3. | no | no | yes | no | yes |
| 4. | no | yes | no | yes | no |
| 5. | yes | no | yes | yes | yes |
| 6. | no | yes | yes | yes | no |
| 7. | no | yes | yes | no | ? |

Use Naive Bayes method to predict whether the last person will have the disease. Be sure to use Laplace smoothing. Show the steps for your calculation.

# 4 K-means Clustering (10 points)

Show 2 iterations of the k-means algorithm (k = 2) on the following one-dimensional data set:

Data: [ 4, 1, 9, 12, 6, 10, 2, 3, 9 ]

First iteration: cluster centers (randomly selected): 1, 6

Data assignment:

Cluster 1: [ 1, 2, 3 ]

Cluster 2: [ 4, 9, 12, 6, 10, 9 ]

(a) What are the cluster centers, and then the data assignments, that would be obtained for each of two more iterations? Show your work.

(b) After your iterations, has the algorithm converged to a solution at this point, or not? How can you tell?

# 5 Image segmentation using clustering (20 points)

You will next test your implementation by applying clustering to segment and recolor an image. Write your code in a script segment.ipynb.

[5 pts] Download the following images: panda and wm. Load them in your program using img=mpimg.imread("panda.jpeg");. This will return a HxWx3 matrix per image, where H and W denote height and width, and the image has three channels (R, G, B). Convert the image from int to double format. To avoid a long run of your code, downsample the images (reduce their size) to size around 100x100.

[5 pts] To perform segmentation, you need a representation for every image pixel. We will use a three-dimensional feature representation for each pixel, consisting of the R, G and B values of each pixel. Use fimg = img.reshape(-1, 3) to convert the 3D matrix into a 2D matrix with pixels as the rows and channels (features) as the columns. Use KMeans function in sklearn, to perform clustering over the pixels of the image.

[5 pts] Then recolor the pixels of each image according to their cluster membership. In particular, replace each pixel with the average R, G, B values for the cluster to which the pixel belongs (i.e. recolor using the cluster means). Show the recolored image using imgplot = plt.imshow(img), but convert it to format uint8 before displaying.

[5 pts] Experiment with different values of number of clusters K = 2,3,4,5. Show the images.

You may start your program as follows:

```
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

img=mpimg.imread("panda.jpeg")
print(img.shape)
# downsample image
img2=img[::8,::14,::1]
print(img2.shape)
# reshape to 2-d
fimg = img2.reshape(-1, 3)
print(fimg.shape)

kmeans = KMeans(n_clusters=2, random_state=0).fit(fimg)
print(kmeans.labels_)
```