Assignment #1

B05902120 / Yu-Ting, TSENG

Oct 4, 2019

Bayes Decision Rule

For a 2-class problem based on a single feature $x \in [0, 6]$, the class PDFs are defined as below:

$$P(x|\omega_1) = \begin{cases} \frac{1}{4}x, & x \in [0, 2) \\ -\frac{1}{4}x + 1, & x \in [2, 4] \\ 0, & \text{otherwise} \end{cases}$$

$$P(x|\omega_2) = \begin{cases} \frac{1}{4}, & x \in (2, 6) \\ 0, & \text{otherwise} \end{cases}$$

Determine the minimum P_e decision scheme with $P(\omega_2) = \frac{4}{9}$. Please state clearly what the decision regions $R1$ and $R2$ are. What is the resulting probability of error P_e?

Let T be decision boundary threshold.

$$\begin{aligned} P_e &= \int_{\omega_1} P(\omega_2|x)P(x)dx + \int_{\omega_2} P(\omega_1|x)P(x)dx \\ &= \int_{\omega_1} P(x|\omega_2)P(\omega_2)dx + \int_{\omega_2} P(x|\omega_1)P(\omega_1)dx \\ &= \int_{T}^{\infty} P(x|\omega_1)P(\omega_1)dx + \int_{-\infty}^{T} P(x|\omega_2)P(\omega_2)dx \\ &= \int_{0}^{2} \frac{1}{4}x * \frac{5}{9}\, dx + \int_{2}^{T} (-\frac{1}{4}x + 1) * \frac{5}{9}\, dx + \int_{T}^{4} \frac{1}{4} * \frac{4}{9}\, dx \\ &= \frac{5}{18} + ((-\frac{5}{72}T^2 + \frac{5}{9}T) - \frac{5}{6}) + (\frac{4}{9} - \frac{1}{9}T) \\ &= -\frac{5}{72}T^2 + \frac{4}{9}T - \frac{1}{9} \end{aligned}$$

To minimize the error, we get $P_e = \frac{1}{2}$ when $T = 2$.

Principle Component Analysis

Principal component analysis (PCA) is a technique for dimensionality reduction which performs a linear mapping of the data to a lower-dimensional space in such a way that the variance of the data in the low-dimensional representation is maximized. In this problem, you will perform PCA on a dataset of face images.

The folder `p2_data` contains face images of 40 different subjects (classes) and 10 grayscale images for each subject, all of size $(56, 46)$ pixels. Note that `i_j.png` is the $j-th$ image of the $i-th$ person, which is denoted as $\textbf{person}_i\textbf{image}_j$ for simplicity.

First, split the dataset into two subsets (i.e., training and testing sets). The first subset contains the first 6 images of each subject, while the second subset contains the remaining images. Thus, a total of $6 * 40 = 240$ images are in the training set, and $4 * 40 = 160$ images in the testing set.

In this problem, you will compute the eigenfaces of the training set, and project face images from both the training and testing sets onto the same feature space with reduced dimension.

- Platform: Unix (Macbook)
- Language: Python 3.6

Perform PCA on the training set. Plot the mean face and the first four eigenfaces.

For mean face, we calculate the mean values of all pixels at the corresponding position via `numpy` conveniently and get the resultant image.

```
res = np.mean(img, axis = 0)
```

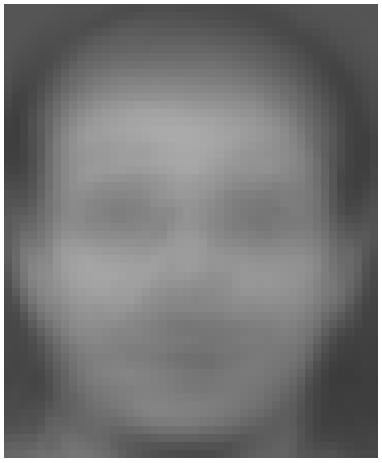

Figure 1: meanFace

For eigen face, we import `PCA`. First calculate the mean values as problem 1 and compute the differences between the original input images and the mean values. Finally, we can get the results by `components_[$index]` and normalize the values.

```
pca  = PCA()
mean = np.mean(img, axis = 0)
diff = img - mean
modl = pca.fit(diff)

res  = modl.components_[k]
res -= np.min(res)
res /= np.max(res)
res *= 255
```

Figure 2: eigenFace_1

Figure 3: eigenFace_2

Figure 4: eigenFace_3

Figure 5: eigenFace_4

Take $person_1 image_1$ you obtained above and project to eigenspace you obtainedabove. Reconstruct this image using the first $n = 3, 45, 140, 229$ eigen faces. Plot the four reconstructed images.

To begin, we could adopt the pca we construct above and project the target image `1_1.png` on the eigenspace. Eventually, we are might be able to get the resultant images by reconstructing it.

```
diff = imgTar - mean
diff = np.reshape(diff, (1, shp[0] * shp[1]))
tran = pca.transform(diff)
res  = (tran[:,:$n] @ modl.components_[:$n]) + mean
```

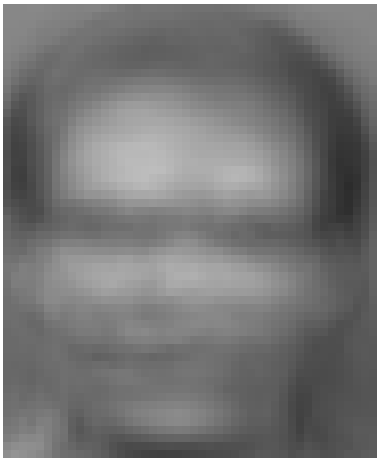

Figure 6: reconstruct_3

Figure 7: reconstruct_45

Figure 8: reconstruct_140

Figure 9: reconstruct_229

For each of the four images you obtained in 2., compute the mean squared error (MSE) between the reconstructed image and the original image. Record the corresponding MSE

values in your report.

We could calculate easily through computation.

```
mse  = np.mean((imgOri - imgTar)**2)
```

	$n = 3$	$n = 45$	$n = 140$	$n = 229$
mse	1007.256938	262.787871	19.096631	0.109632

Now, apply the k-nearest neighbors algorithm to classify the testing set images. First, you will need to determine the best k and n values by 3-fold cross-validation. For simplicity, the choices for such hyperparameters are $k = 1, 3, 5$ and $n = 3, 45, 140$. Show the cross-validation results and explain your choice for (k, n).

There is a useful library `KNeighborsClassifier`. We can adopt the library simply as below:

```
knn = KNeighborsClassifier(n_neighbors = $k)
knn.fit(tran[:,:$n], labTrn)
```

`GridSearchCV` is also a useful tools, we could get the best results based on various `k` and `n` of k-fold input data.

```
par = $n_list
clf = GridSearchCV(knn, par, cv = 3)
```

The following table records the results:

	$n = 3$	$n = 45$	$n = 140$
$k = 1$	0.70416667	0.92916667	0.92916667
$k = 3$	0.61666667	0.85833333	0.85833333
$k = 5$	0.52083333	0.79166667	0.75416667

It is obviously we should choose 1 for k and 45 for n. In this case, we predict the results of training data by the knn classifier we have trained above. At last, we compute the accuracy and get the number 0.956250.

Visual Bag-of-Words

A bag-of-words model (BoW) can be applied to image classification, by treating image features as words. In computer vision, a bag of visual words is a vector of occurrence counts of a vocabulary of local image features. In this problem, you will implement a basic image-based BoW model for an image dataset with 4 categories, where we use small

image patches as features.

The folder `p3_data` contains images of 4 categories (classes) and 500 RGB images for each category, all of size (64, 64, 3) pixels. First, split the dataset into two subsets (i.e., training and testing sets). The first subset contains the first 375 images of each category, while the second subset contains the remaining images. Thus, a total of $375 * 4 = 1500$ images are in the training set, and $125 * 4 = 500$ images in the testing set. We will denote them as `X_train` and `X_test`, respectively.

- Platform: Unix (Macbook)
- Language: Python 3.6

Divide up each image in both `X_train` and `X_test` into a grid of (16,16,3) image patches. Since each image is of size (64, 64, 3), this will result in 16 different patches of size (16, 16, 3) for each image. You can imagine the patches as puzzle pieces which together would reconstitute the whole image. Pick 4 images (one from each category) randomly and plot 3 such patches from each image you choose. Describe whether you are able to classify an image by seeing just a few patches and write why.

I think it is hard to classify an image by seeing only a few patches. For us, as human, we can hardly find the unique features of those patches. For computer, it is even much harder to extract features; I have take use of the python tools, SIFT and ORB, but both of them could not find any interest points. However, the frequency of the pixel values might be helpful. The symbolic colors of the label selected in this assignment are totally different, therefore I assume that the results might not be so bad; on the other hand, they would not have great performance.

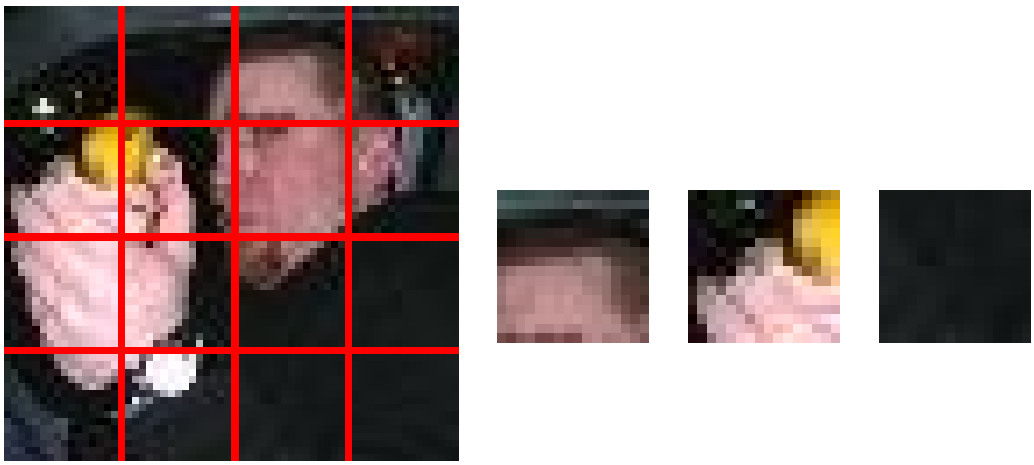

Figure 10: patchesBanana

Figure 11: patchesFountain

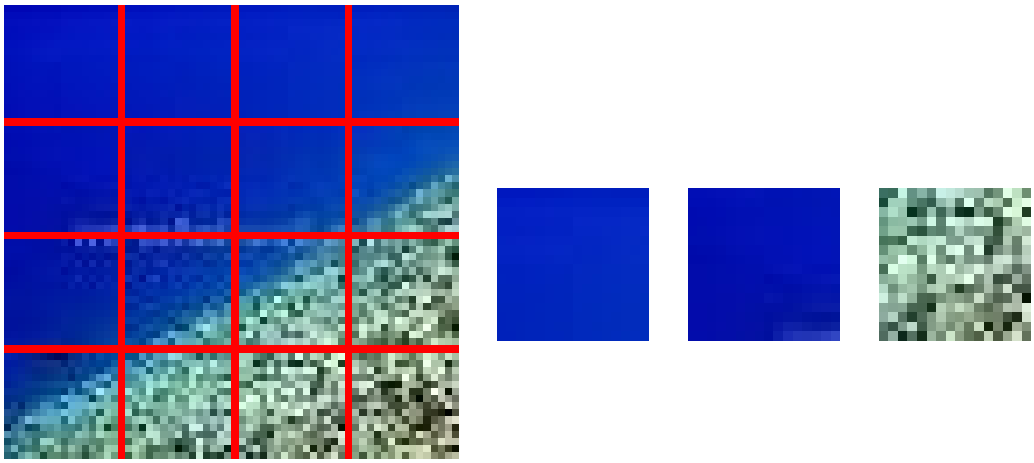

Figure 12: patchesReef

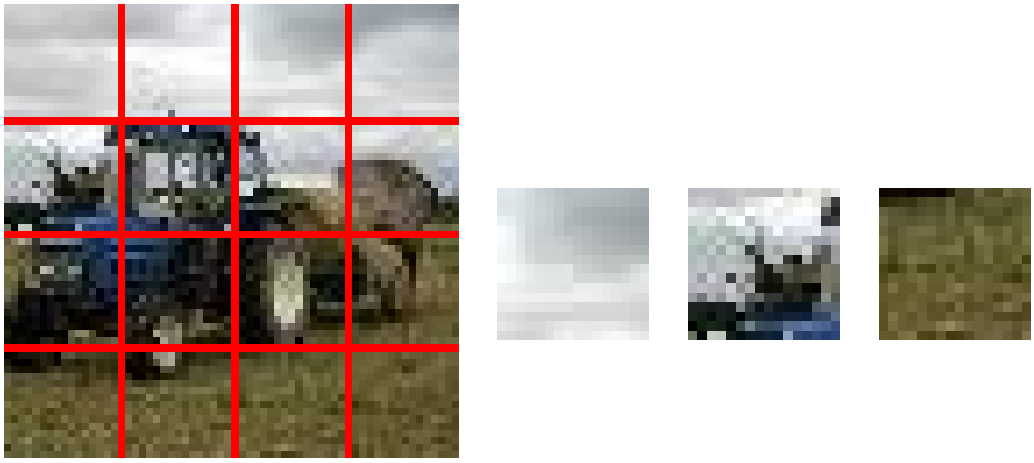

Figure 13: patchesTractor

Flatten the patches into 1D vectors and store them as variables `X_train_patches` and `X_test_patches`. Since each patch is of size $16 * 16 * 3 = 768$ where each image contains 16 such patches, `X_train_patches` and `X_test_patches` should be of size

(24000, 768) and (8000, 768), respectively.

Use the k-means algorithm to divide the training patches (features) into C clusters. You may choose $C = 15$ and maximum number of iterations = 5000 for simplicity. The centroid of each cluster then indicates a visual word.

Construct the 3-dimensional PCA subspace from the training features. Randomly select 6 clusters from the above results. Plot the visual words (i.e., centroids) and their associated features (i.e., patches) in this PCA subspace. Use the same color for features belonging to the same cluster and the same color for the 6 centroids in your visualization.

I take use of a tool defined in `sklearn` and find out the centroids of 15 divided clusters.

```
kmn = KMeans(n_clusters = 15)
ret = kmn.fit_predict(pat)
cen = kmn.cluster_centers_
```

Next, I plat a graph in 3D with various colors by using `PCA` ,which is mentioned above, and set the dimension to be 3. Nevertheless, I find out that it is hard to observe where the centroids are. To solve this problem, I plot another three graphs which is tangent plane of different axis. Each black crosses are the centroids of the clusters.

```
pca = PCA(n_components = 3)
modl = pca.fit_transform(pat, lab)
proj = pca.transform(cen)
```

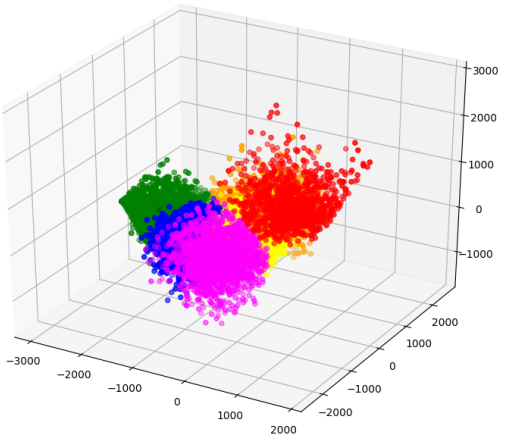

Figure 14: 3 dimension

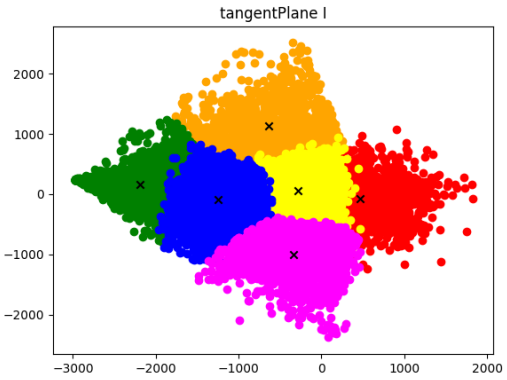

Figure 15: tangentPlane_1

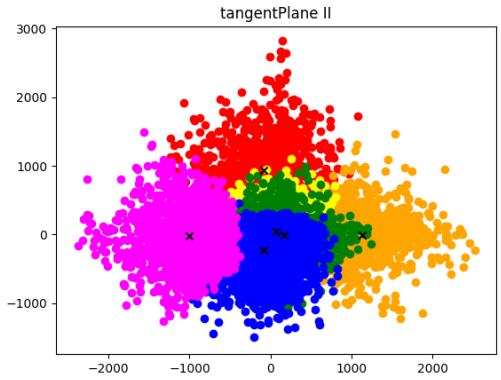

Figure 16: tangentPlane_2

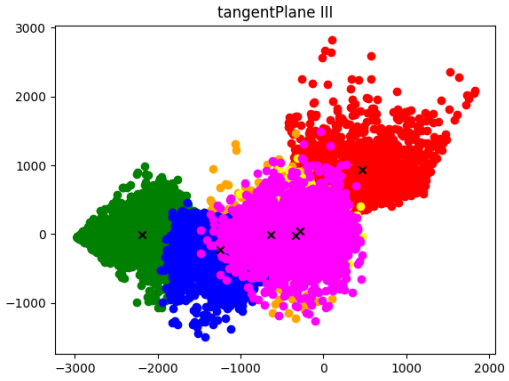

Figure 17: tangentPlane_3

With the derived dictionary of visual words, you can now represent each image as BoW features. We will adopt the soft-max strategy when encoding image patches as detailed below.

Take an image with 4 patches and a learned dictionary with 3 visual words ($C = 3$) for example. Table 1 lists the Euclidean distance between the patches f_i (with $i = 1, 2, 3, 4$) and the centroids c_j (with j = 1, 2, 3). For each patch, the reciprocal of its distance to each centroid would be normalized. Each attribute in the BoW is then determined by the maximum value of the softencoded features in that dimension (i.e., max-pooling). For example, the BoW of the patches in Table 1 would be [0.55 0.27 0.55] by taking the maximum value of each column.

Now, compute the BoW of training images in `X_train` with $C = 15$, resulting in a matrix of size (1500, 15). Choose one image from each category and visualize its BoW using histogram plot. `numpy` has a lot of function for common used mathematics formula, including `np.linalg.norm` and `np.reciprocal`.

```
arrTrn[i][j] = np.linalg.norm(patTrn - cen[j])
arrTrn[i][j] = np.reciprocal(arrTrn[i][j])
arrTrn[i] = arrTrn[i] / np.linalg.norm(arrTrn[i])
bow[k] = np.max(arrTrn, axis = 0)
```

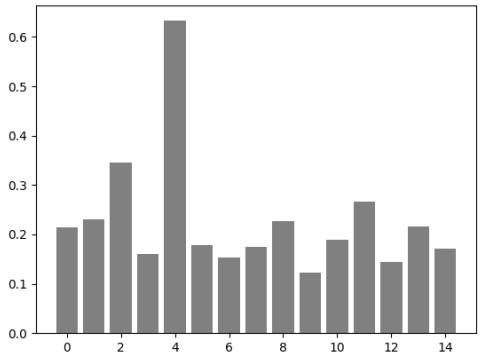

Figure 18: bagOfWords_1

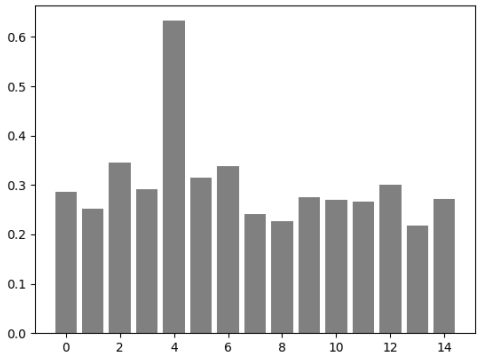

Figure 19: bagOfWords_2

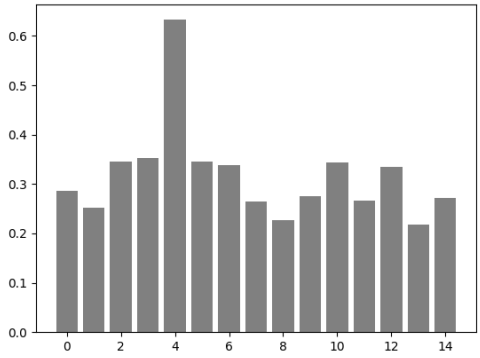

Figure 20: bagOfWords_3

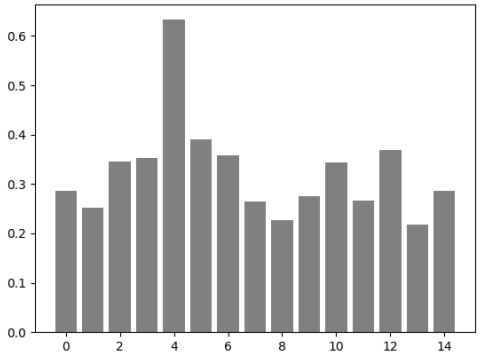

Figure 21: bagOfWords_4

Adopt the k-nearest neighbors classifier (k-NN) to perform classification on `X_test` using the above BoW features (you may choose $k = 5$ for simplicity). Report the classification accuracy.

```
prodigy1026 at TSENG-MacBook in ~/Downloads/hw1 executing
> python3 Problem3.py
[Done] Dividing the data!
[Done] Implementing k-means algorithm!
[Done] Computing bagOfWords of training images!
0.466
```