Machine Learning for Mechanical Engineering

Homework 4

Due Monday, 04/09/2018, 9:30 AM

by Prof. Seungchul Lee
Industrial AI Lab
POSTECH

- For your handwritten solution, hand in your hardcopy at class.
- For your code, email your .ipynb file to (`iai.postech@gmail.com`)
- When you send an e-mail, write down [Machine Learning HW4] on the title
- And please write your NAME on your .ipynb files. ex) 김지원_20182315_HW2.ipynb
- Do not submit a printed version of your code. It will not be graded.

Problem 1

In this problem, we will try to classify handwritten digits. For the sake of simplicity, we simplify this digit classification problem into a binary classification between digit 0 and digit 1.

You can download images from this link (https://www.dropbox.com/sh/2s154nsafde9nra/AACeAK_EwjoclWScP7O1U-5la?dl=0).

Step 1. Load data

You can load the digit images using the below code. Each dataset contains 500 gray-scaled images with size of (28, 28).

In [1]:

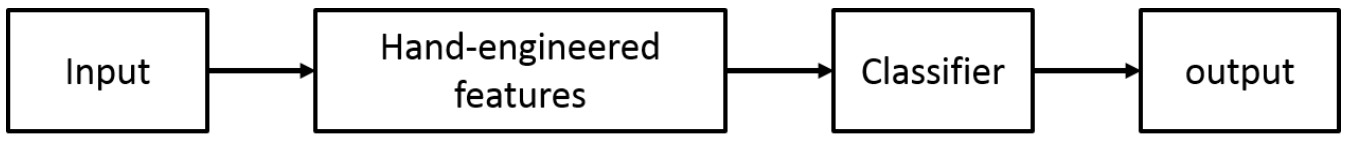
```
from six.moves import cPickle
digit0 = cPickle.load(open('./data/digit0.pkl', 'rb'))
digit1 = cPickle.load(open('./data/digit1.pkl', 'rb'))
```

(a) Check the shape of each dataset. Randomly pick images from each datasets and `imshow` the images.

In [2]:

```
# Your code here
```

Step 2. Extract features



(b) Write the code for feature extraction. We will construct the feature vectors for each dataset with the following three features :

- The average value of pixels located at the center of the image ([9:18, 9:18])
- The average value of pixels over the entire location
- Include the ones as the bias term

(c) Then you will have (500, 3) feature vectors for each dataset. Plot the first two features for every samples on the 2D-plane.

In [3]:

```
# Your code here
```

Problem 2

We will apply perceptron to the feature vectors defined in Problem 1.

(a) The below is a pseudo code of the perceptron. Write the code to classify the digits using the given perceptron pseudo code.

```
For k = 1:100 {
      For j = 1:100 {
            i = random integer selection between 1 and 1000
            compute yhat(i)
            if yhat(i) is wrong {
                w = w + y(i)x(i)
            }
    }
}
```

In [4]:

```
# Your code here
```

(b) Plot the decision boundary of your classifier on the figure you have plotted in Step 2 of Problem 1.

In [5]:

```
# Your code here
```

Problem 3

We will apply SVM to the feature vectors defined in Problem 1.

(a) Write the code to classify the digits with SVM using cvxpy.

In [6]:

```
# Your code here
```

(b) Plot the decision boundary of your classifier on the figure you have plotted in Step 2 of Problem 1.

In [7]:

```
# Your code here
```

Problem 4

(a) Let $f(x) = \omega^T x$ be a linear boundary of a binary classifier and $y \in \{-1, 1\}$. Explain $y \cdot f(x) \leq 0$ when the classifier is wrong and $y \cdot f(x) > 0$ when the classifier is correct.

(b) Let $\ell(f(x), y)$ be a loss function. Ideally, the loss function for the classification problem is $u(-y \cdot f(x))$ where $u(\cdot)$ is a step function. Explain the meaning of $J = \sum_{i=1}^{N} u(-y_i \cdot f(x_i))$.

(c) Explain why the ideal loss function cannot be implemented when we want to use a gradient descent algorithm?

(d) In the logistic regression, we define

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Then, show

$$\frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z))$$

Note: this property is useful in a gradient descent for a parameter learning.

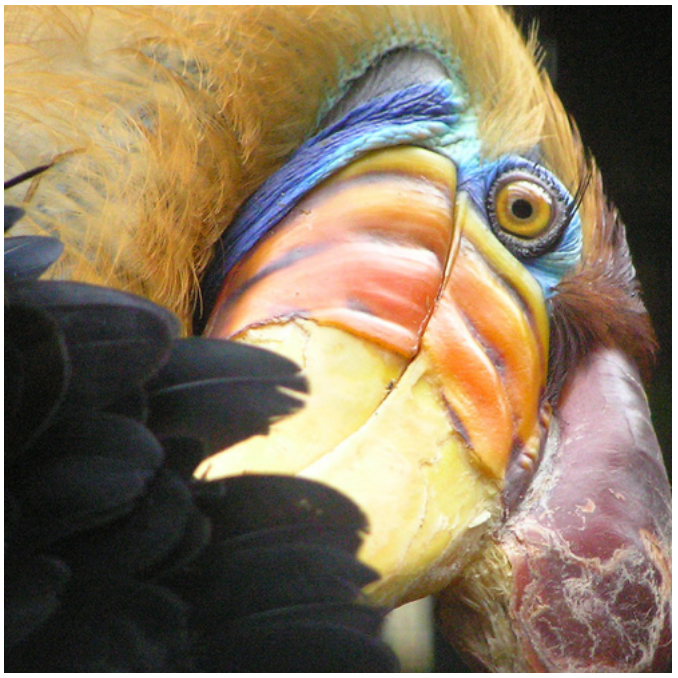
(e) Plot the ideal loss function and the loss function of the logistic regression. Compare the shape of both loss functions and explain why the logistic regression can approximately solve the classification problem.

Problem 5

You will use K-means to compress an image by reducing the number of colors it contains. You can download images from this link (https://www.dropbox.com/sh/jv300h0z9a93evb/AACMbKwcAg6tIjo1JIn6u3i7a?dl=0).

(a) Image Representation

The data for this exercise contains a 128-pixel by 128-pixel TIFF image named "bird.tiff." It looks like the picture in Figure 1.



In a straightforward 24-bit color representation of this image, each pixel is represented as three 8-bit numbers (ranging from 0 to 255) that specify red, green and blue (RGB) intensity values. Our bird photo contains thousands of colors, but we'd like to reduce that number to 16. By making this reduction, it would be possible to represent the photo in a more efficient way by storing only the RGB values of the 16 colors present in the image.

In this problem, you will use K-means to reduce the color count to $k = 16$. That is, you will compute 16 colors as the cluster centroids and replace each pixel in the image with its nearest cluster centroid color.

(b) K-means in Python

You can load a bird image using the following code.

In [8]:

```
from six.moves import cPickle
import matplotlib.pyplot as plt

A = cPickle.load(open('./data/bird.pkl', 'rb'))

plt.figure(figsize=(4, 4))
plt.imshow(A.astype('uint8'))
plt.axis('off')
plt.show()

print('Matrix shape: {}'.format(A.shape))
```

```
<Figure size 400x400 with 1 Axes>
```

```
Matrix shape: (128, 128, 3)
```

This creates a three-dimensional matrix A whose first two indices identify a pixel position and whose last index represents red, green, or blue. For example, $A(50, 33, 3)$ gives you the blue intensity of the pixel at position $y = 50, x = 33$. (The y-position is given first, but this does not matter so much in our example because the x and y dimensions have the same size).

Your task is to compute 16 cluster centroids from this image, with each centroid being a vector of length three that holds a set of RGB values. Here is the K-means algorithm as it applies to this problem:

(c) K-means algorithm

1. For initialization, sample 16 colors randomly from the original picture. There are your k means $\mu_1, \mu_2, \cdots, \mu_k$.

2. Go through each pixel in the small image and calculate its nearest mean.

$$c^{(i)} = \arg min_j \|x^{(i)} - \mu_j\|^2$$

3. Update the values of the means based on the pixels assigned to them.

$$\mu_j = \frac{\sum_i^m 1\{c^{(i)} = j\} x^{(i)}}{\sum_i^m 1\{c^{(i)} = j\}}$$

4. Repeat steps 2 and 3 until convergence. This should take between 30 and 100 iterations. You can either run the loop for a preset maximum number of iterations, or you can decide to terminate the loop when the locations of the means are no longer changing by a significant amount.

Note: In Step 3, you should update a mean only if there are pixels assigned to it. Otherwise, you will see a divide-by-zero error. For example, it's possible that during initialization, two of the means will be initialized to the same color (*i.e.* black). Depending on your implementation, all of the pixels in the photo that are closest to that color may get assigned to one of the means, leaving the other mean with no assigned pixels.

When you have recalculated the image, you can display it. When you are finished, compare your image to the one in the solutions.

In [9]:

```
# Your code here
```