Project 2 Image Classification by Maximum-A-Posterior Decision

Fall 2018
(Due: Nov 2, 2018)

Project is due at 4:30pm. Please put your homework in the dropbox located at MSEE 330. No late homework will be accepted.

1 Introduction

Image classification is an important problem in computer vision and (probably) the most widely used test bed problem in artificial intelligence. In this project, we are going to study a simplified version of a common image classification problem by classifying foreground and background regions in an image. This may sound challenging as we have not taught these in class. However, the tools you need are actually ready, e.g., conditional probability, Bayes’ rule, normal distribution.

The image that we are going to classify consists of a cat and some grass. The size of this image is $500 \times 375$ pixels. The left hand side of Figure 1 shows the image, and the right hand side of Figure 1 shows a manually labeled “ground truth”. Your task is to do as much as you can to extract the cat from the grass.

Figure 1: The “Cat and Grass” image.

1.1 Background Python Commands

So how would you attempt the problem? Well, the first thing you need to learn is how to read images into Python variables, if you have not learned yet. On the course website, you will see a data.zip. Download this .zip file and decompress it into your current working directory. You will be able to see an image file cat_grass.jpg. Read this image using

$$Y = \text{plt.imread('cat_grass.jpg')} / 255$$

The function plt.imread reads the image file and stores it as an array. We divide the whole array by 255 to convert the into a floating point array within range $(0,1)$. Such conversion is necessary because the training was done in such range.

Next, we need to understand how to manipulate these pixels. To access to the $(i, j)$th pixel of the image, you can type $Y[i,j]$. But you have to be careful, because Python’s indexing goes with the order “row-column” (i.e., the matrix indexing), not the conventional $(x,y)$ coordinate. Therefore, $Y(4,2)$ means the 4th row and the 2nd column, i.e., vertically the 4th pixels and horizontally the 2nd pixels.

---

1 Image Source: [http://www.robots.ox.ac.uk/vgg/data/pets/](http://www.robots.ox.ac.uk/vgg/data/pets/)

© 2018 Stanley Chan. All Rights Reserved.
Beside pixel indexing, we also need to understand how to extract a patch from an image. By patch we meant a small neighborhood surrounding the pixel. For example, when we say extract an 8 × 8 patch at pixel \((i, j)\), we mean that we want to extract \(Y[i:i+8,\ j:j+8]\) from the image. If you like to repeatedly extract patches at every pixel, you can write a double for-loop:

```python
for i in range(M-8):
    for j in range(N-8):
        z = Y[i:i+8, j:j+8]
        ... % other steps; to be discussed
end
end
```

Here, \(M\) and \(N\) are the number of rows and columns of the image, respectively.

Remark 1: If you have a patch \(z = Y[i:i+8, j:j+8]\), you will see that the size of \(z\) is 8 × 8. To convert \(z\) into a 64 × 1 column vector, you can type

```python
>> z_vector = z.flatten('F');
```

To convert \(z\_vector\) back to a patch, you can type

```python
>> z = reshape(z_vector, (8, 8), order = 'F');
```

Remark 2: By performing the above procedure, we will left out the boundary pixels and we made the top left pixel of a patch as its “center”. Of course, these can be fixed (how?) but we will not drill into them.

### 1.2 Statistical Commands in Python

In `data.zip`, you will also find `training_cat.txt` and `training_grass.txt` which is the training data for this project. Load the data using `train_cat = np.matrix(np.loadtxt('train_cat.txt', delimiter = ','))`. The data need to be stored as matrix for later use.

The sizes of the arrays are 64 × \(K\), where \(K\) corresponds to the number of training samples and 64 corresponds to the size of the block 8 × 8. In this project, you will need to compute the mean vectors and the covariance matrices of these data arrays. What are mean vectors and covariance matrices? For any collection of vectors, e.g., \(\{x_1^{(cat)}, \ldots, x_K^{(cat)}\}\), the mean vector and the covariance matrix are

\[
\mu_{(cat)} = \frac{1}{K} \sum_{k=1}^{K} x_k^{(cat)}, \quad \text{and} \quad \Sigma_{(cat)} = \frac{1}{K} \sum_{k=1}^{K} (x_k^{(cat)} - \mu_{(cat)})(x_k^{(cat)} - \mu_{(cat)})^T. \tag{1}
\]

In Python, these can be computed easily via the `numpy.mean(x, 1)` and `numpy.cov(x)` commands.

Hint: The size of \(\mu_{(cat)}\) should be 64 × 1 and the size of \(\Sigma_{(cat)}\) should be 64 × 64.

## 2 Maximum-A-Posteriori (MAP) Decision

Now let us discuss how to perform the classification using the training data. The method we will use is called maximum-a-posteriori (MAP) decision.

### 2.1 Multivariate Gaussian

In statistical learning, the first thing we need to do is to give some model about the data. For the sake of this project, let us choose the Gaussian distribution. No other reasons, just make it simple.

Now, let us look at the data. Our data is a collection of 8 × 8 patches (or equivalently 64 × 1 vectors). Therefore, we have to use a multivariate Gaussian instead of the single variable Gaussian. Multivariate Gaussian is nothing but some generalization of the single variable Gaussian. If you like to know more about the multi-variate Gaussian, you can read textbook chapter 6.
Here are more discussions about a multivariate Gaussian. Let \( \mathbf{Z} \) be a vector random variable representing a patch taken from the image “Cat and Grass”. We assume that the conditional distribution of \( \mathbf{Z} \) given its class label follows the distribution:

\[
\begin{align*}
  f_{\mathbf{Z} | \text{class}}(\mathbf{z} | \text{cat}) &= \frac{1}{(2\pi)^{d/2}|\Sigma_{(\text{cat})}|^{1/2}} \exp \left\{ -\frac{1}{2}(\mathbf{z} - \mu_{(\text{cat})})^T \Sigma_{(\text{cat})}^{-1}(\mathbf{z} - \mu_{(\text{cat})}) \right\}, \\
  f_{\mathbf{Z} | \text{class}}(\mathbf{z} | \text{grass}) &= \frac{1}{(2\pi)^{d/2}|\Sigma_{(\text{grass})}|^{1/2}} \exp \left\{ -\frac{1}{2}(\mathbf{z} - \mu_{(\text{grass})})^T \Sigma_{(\text{grass})}^{-1}(\mathbf{z} - \mu_{(\text{grass})}) \right\}.
\end{align*}
\]

These equations may look complicated to you in a first glance, but they are just the high-dimensional extension of the single variable Gaussian we learned in class — if \( \mu \) is a scalar \( \mu \), and \( \Sigma \) is also a scalar \( \sigma^2 \), then we will obtain the one-dimensional Gaussian.

A few more words about the notations to help you understand these two equations. Here, the constant \( d \) is the dimensionality of the data, which is \( d = 64 \) in our case. \( \Sigma_{(\text{cat})} \) is the covariance matrix of the class “cat”, and it has the size \( 64 \times 64 \). The determinant of this matrix is denoted as \( |\Sigma_{(\text{cat})}| \), and can be calculated in Python using the command \texttt{numpy.linalg.det}. The inverse of this matrix is denoted as \( \Sigma_{(\text{cat})}^{-1} \). In Python, you can use the command \texttt{numpy.linalg.pinv}.

So what is the meaning of \( f_{\mathbf{Z} | \text{class}}(\mathbf{z} | \text{cat}) \)? This conditional probability specifies the likelihood — if we knew that a patch \( \mathbf{Z} \) belongs to “cat”, what is the probability that this random variable \( \mathbf{Z} \) would take a particular realization \( \mathbf{z} \)? By using the multivariate Gaussian, we assume that this conditional probability follows the Gaussian distribution.

### 2.2 Posterior Distributions

Given a patch \( \mathbf{Z} \), how would you decide whether it belongs to “cat” or “grass”? From the discussion above, it is tempting to compare \( f_{\mathbf{Z} | \text{class}}(\mathbf{z} | \text{cat}) > f_{\mathbf{Z} | \text{class}}(\mathbf{z} | \text{grass}) \). However, comparing these conditional probabilities is not what we want: it is the probability that we see a patch \( \mathbf{Z} \) given the class label, not the class label given the patch \( \mathbf{Z} \). What we should really compare is the posterior probability

\[
f_{\text{class} | \mathbf{Z}}(\text{cat} | \mathbf{z}) \geq f_{\text{class} | \mathbf{Z}}(\text{grass} | \mathbf{z}).
\]

The posterior probabilities are related to the conditional probabilities via the Bayes rule as

\[
\begin{align*}
  f_{\text{class} | \mathbf{Z}}(\text{cat} | \mathbf{z}) &= \frac{f_{\mathbf{Z} | \text{class}}(\mathbf{z} | \text{cat}) f_{\text{class}}(\text{cat})}{f_{\mathbf{Z}}(\mathbf{z})}, \\
  f_{\text{class} | \mathbf{Z}}(\text{grass} | \mathbf{z}) &= \frac{f_{\mathbf{Z} | \text{class}}(\mathbf{z} | \text{grass}) f_{\text{class}}(\text{grass})}{f_{\mathbf{Z}}(\mathbf{z})},
\end{align*}
\]

The distributions \( f_{\text{class}}(\text{cat}) \) and \( f_{\text{class}}(\text{grass}) \) are called the prior distributions. Since we only have two classes, the prior distribution is a discrete probability mass function. For simplicity we assume that \( f_{\text{class}}(\text{cat}) \) and \( f_{\text{class}}(\text{grass}) \) are the ratios between the number of samples in the training set:

\[
\begin{align*}
  f_{\text{class}}(\text{cat}) &= \frac{K_{(\text{cat})}}{K_{(\text{cat})} + K_{(\text{grass})}}, \\
  f_{\text{class}}(\text{grass}) &= \frac{K_{(\text{grass})}}{K_{(\text{cat})} + K_{(\text{grass})}},
\end{align*}
\]

where \( K_{(\text{cat})} \) is the number of training samples in \texttt{train.cat}, and \( K_{(\text{grass})} \) is the number of training samples in \texttt{train.grass}. Substituting these equations into (4) yields

\[
f_{\mathbf{Z} | \text{class}}(\mathbf{z} | \text{cat}) f_{\text{class}}(\text{cat}) \geq f_{\mathbf{Z} | \text{class}}(\mathbf{z} | \text{grass}) f_{\text{class}}(\text{grass}),
\]

where we canceled out the common factor \( f_{\mathbf{Z}}(\mathbf{z}) \). The decision rule based on (9) is called the Maximum-a-Posteriori (MAP) decision.

The main routine of your program should look like the following.

© 2018 Stanley Chan. All Rights Reserved.
Output = np.zeros((M-8,N-8))  
for i in range(M-8):  
    for j in range(N-8):  
        z = Y(i:i+8, j:j+8)  
        % compute f(z | cat)  
        % compute f(z | grass)  
        % if f(cat | z) > f(grass | z), then set Output(i,j) = 1  

Your task is to fill in these missing blanks.

3 Exercise

1. (10 points) Training. Follow the instructions in Section 1, write a Python function to compute the training outputs.

   def my_training(train_cat, train_grass):  
       % train_cat and train_grass can be loaded from train_cat.txt and train_grass.txt  
       mu_cat = ...  
       mu_grass = ...  
       Sigma_cat = ...  
       Sigma_grass = ...  
       return mu_cat, mu_grass, Sigma_cat, Sigma_grass

   Submit your Python program.

2. (50 points) Testing. Follow the instructions in Section 2, write a MATLAB function for the testing problems.

   def my_testing(Y, mu_cat, mu_grass, Sigma_cat, Sigma_grass, K_cat, K_grass)  
       ...  
       return output

   Submit your Python program. Note: Think about how to speed up the computation. My implementation takes about 11 seconds.

3. (20 points) Results. Show the result of Output using the command

   plt.imshow(output*255, cmap = 'gray')

   and report your runtime using the command

   import time

   ...

   start_time = time.time()
   Output = my_testing(Y, mu_cat, mu_grass, Sigma_cat, Sigma_grass)
   print('My runtime is %s seconds' % (time.time() - start_time))

   Calculate the mean absolute error of your estimate compared to the ground truth. Mean absolute error between an estimate $x$ and the ground truth $x^*$ is defined as

   $$\text{MAE}(x, x^*) = \frac{1}{N} \sum_{n=1}^{N} |x_n - x_n^*|.$$  

4. (20 points) Improvement. This is an open-ended question. Propose a method to reduce the mean absolute error. Show your code and show your result.