

## Homework 4

In this homework, we will implement a Naive Bayes classifier and a KNN classifier.

You will be given two training datasets: `hw5_grass.txt` and `hw5_cat.txt`. Each dataset is stored as a data matrix of size  $64 \times 500$ . Let  $\mathbf{x}_k^{(\text{cat})}$  be the  $k$ -th column of the `cat` matrix. The size of  $\mathbf{x}_k^{(\text{cat})}$  is  $64 \times 1$ , and it represents a *feature vector* of a training sample. There are 500 of these training samples in each dataset.

In your repository, you will also find an image `cat_grass.png`. Load this image into Python using `matplotlib.pyplot`'s command `imread`. For your convenience, we have written a data pre-processing function `preprocess_data` in `helper.py` which you can input the image and extract the feature vector for every pixel. The size of the pre-processed data is  $64 \times 187500$ . Each column of this matrix represents the feature vector of a pixel, and there are 187500 pixels in the image. Let's denote this matrix as  $\mathbf{Y}$ .

### Exercise 1. NAIVE BAYES

In this part of the exercise, we will build a Naive Bayes classifier. For simplicity we will use a multi-dimensional Gaussian model. To construct the model, we need to know the mean vector and the covariance matrix of each class. These can be computed using `numpy.mean` and `numpy.cov` command. (Note that since this time you will be calculating the mean of each row, you should use `numpy.mean(x, 1)`.) If you do everything correct, you will have two vectors  $\boldsymbol{\mu}^{(\text{cat})} \in \mathbb{R}^{64 \times 1}$  and  $\boldsymbol{\mu}^{(\text{grass})} \in \mathbb{R}^{64 \times 1}$ , and two matrices  $\boldsymbol{\Sigma}^{(\text{cat})} \in \mathbb{R}^{64 \times 64}$  and  $\boldsymbol{\Sigma}^{(\text{grass})} \in \mathbb{R}^{64 \times 64}$ .

Consider the  $k$ -th pixel of the image, or correspondingly  $\mathbf{y}_k$ , the  $k$ -th column of the feature matrix  $\mathbf{Y}$ . We will say that this pixel should be **foreground** if

$$\mathcal{N}(\mathbf{y}_k | \boldsymbol{\mu}^{(\text{cat})}, \boldsymbol{\Sigma}^{(\text{cat})}) \mathbb{P}[\text{Class Cat}] \geq \mathcal{N}(\mathbf{y}_k | \boldsymbol{\mu}^{(\text{grass})}, \boldsymbol{\Sigma}^{(\text{grass})}) \mathbb{P}[\text{Class Grass}], \quad (1)$$

and **background** if

$$\mathcal{N}(\mathbf{y}_k | \boldsymbol{\mu}^{(\text{cat})}, \boldsymbol{\Sigma}^{(\text{cat})}) \mathbb{P}[\text{Class Cat}] < \mathcal{N}(\mathbf{y}_k | \boldsymbol{\mu}^{(\text{grass})}, \boldsymbol{\Sigma}^{(\text{grass})}) \mathbb{P}[\text{Class Grass}], \quad (2)$$

where

$$\mathcal{N}(\mathbf{y}_k | \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2} |\boldsymbol{\Sigma}|^{1/2}} \exp \left\{ -\frac{1}{2} (\mathbf{y}_k - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{y}_k - \boldsymbol{\mu}) \right\}.$$

In this equation,  $d$  is the number of features per point (64, in this case).

You can take log on both sides of Equation (1) and (2) to bypass the exponential computation.

$$\log(\mathcal{N}(\mathbf{y}_k | \boldsymbol{\mu}, \boldsymbol{\Sigma}) \mathbb{P}[\text{Class}]) = -\frac{d}{2} \log(2\pi) - \frac{1}{2} \log(|\boldsymbol{\Sigma}|) - \frac{1}{2} (\mathbf{y}_k - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{y}_k - \boldsymbol{\mu}) + \log(\mathbb{P}[\text{Class}]).$$

In this exercise, we can assume that  $\mathbb{P}[\text{Class Grass}] = \mathbb{P}[\text{Class Cat}] = \frac{1}{2}$ . Note that  $|\boldsymbol{\Sigma}|$  denotes the determinant of  $\boldsymbol{\Sigma}$  which can be computed by `numpy.linalg.det` command.

Note, also, that the first term in this equation is the same for both classes (because  $d$  is the same for both classes), so you do not need to compute it. The other two terms are different for the two classes:  $\boldsymbol{\Sigma}$  is the covariance matrix for the specific class you care about.

For each of the 187500 pixels of the feature matrix  $\mathbf{Y}$ , decide which pixels should be classified as foreground and which pixel should be classified as background. Mark the foreground as 1 and background as 0. Save your result as a  $187500 \times 1$  vector `output` and then reshape it to  $375 \times 500$  using the command `numpy.reshape`. Finally save the image using `matplotlib.pyplot.imsave`.

Hints:

1. You may want to precompute the quantity  $\boldsymbol{\Sigma}^{-1}$  to save time.

## Exercise 2. KNN

We will now write a KNN classifier for the same dataset. To do so, we pick the  $k$ -th column of the  $\mathbf{Y}$  feature matrix. Call this column vector as  $\mathbf{y}_k \in \mathbb{R}^{64 \times 1}$ .

For each class **cat** and **grass**, compute the distance between  $\mathbf{y}_k$  and the data points in the datasets. That is,

$$d(\mathbf{y}_k, \mathbf{x}_j^{(\text{cat})}) = \|\mathbf{y}_k - \mathbf{x}_j^{(\text{cat})}\|^2,$$
$$d(\mathbf{y}_k, \mathbf{x}_j^{(\text{grass})}) = \|\mathbf{y}_k - \mathbf{x}_j^{(\text{grass})}\|^2,$$

where  $j = 1, \dots, 500$ . Then, pick the  $K = 5$  nearest neighbors by inspecting the distances. (Hint: `numpy.argsort` will return the indices that would sort an array in ascending order.) If there are more samples labeled as **cat**, then return the output as foreground; otherwise as background.

Write a for-loop to loop through all the 187500 pixels of the feature matrix  $\mathbf{Y}$  and decide which pixels should be classified as foreground and which pixel should be classified as background. Mark the foreground as 1 and background as 0. Save your result as a  $187500 \times 1$  vector `output` and then reshape it to  $375 \times 500$  using the command `numpy.reshape`. Finally save the image using `matplotlib.pyplot.imshow`.

Hint: For a given test point, you may be tempted to loop through all of the training points and compute the distance to each. This means that you will have a doubly-nested loop (187500 testing points in the outer loop, and for each of those points, looping through 1000 training points). This will work, but it will be very slow. You can use `numpy` matrix routines, and they way they automatically expand vectors into matrices, to compute the distances to all the training points much more quickly:

Suppose  $\mathbf{T}$  is a matrix containing your training points (it should be  $64 \times 1000$ ), and  $\mathbf{y}$  is your testing point (a  $64 \times 1$  vector).

$\mathbf{D} = \mathbf{T} - \mathbf{y}$  will create a new  $64 \times 1000$  matrix, where each column is the difference between that training point and  $\mathbf{y}$ .

$\mathbf{D2} = \text{np.power}(\mathbf{D}, 2)$  will square each individual entry in the matrix.

$\mathbf{dists} = \text{np.sum}(\mathbf{D2}, 0)$  will create a new  $1 \times 1000$  matrix where each entry is the sum of the squares of the differences.

Finally, look at the routine `numpy.argsort` to figure out how to create a list of indices sorted by the value of each index in `dists`. (Last hint: this will all work best if the first 500 columns in  $\mathbf{T}$  come from cat, and the second 500 columns in  $\mathbf{T}$  come from grass.)