Assignment: Supervised learning for digit classification

Erik G. Learned-Miller and Cheni Chadowitz

March 6, 2014

Due: Friday March 14, 2014 by 11:59pm by email to the TA

In this assignment, you will explore both the K-Nearest Neighbors classifier and the *maximum a posteriori* (MAP) classifier for handwritten digits. You will use the supervised learning paradigm, in which you are given some examples of two different classes: handwritten 3's and handwritten 5's.

First, you will explore a few variations on the simple K-Nearest Neighbors classifier covered in lecture by implementing a weighting scheme. Second, you will compute the mutual information and use it to select optimal feature pixels for classification.

1. **Setup**

   (a) Download the data file digits.mat:
   `http://people.cs.umass.edu/~elm/Teaching/Data/digits.mat`
   Put the file in your current working directory. Use the command 'load digits.mat' to load it into matlab. Type 'whos' to see the variables that are defined in it. You should see four variables named train_threes, train_fives, test_threes, and test_fives. They are each a group of 50 images stored in a three-dimensional array. Try plotting a few of the images using imagesc to make sure they appear as you expect.

   (b) Download the starter code:
   `http://people.cs.umass.edu/~cheni/370/CS370-Assignment3-StarterCode.zip`
   and place it in your current working directory. You should be able to run the provided function **knnStandard** by passing in the training and test data for both the threes and fives (see the source code for more details, or run "help knnStandard"). It will output an accuracy of 0.90 for all the test images as specified in the comments (and help).

2. **K-Nearest Neighbor** In this part, we explore the effects of using a weighting scheme in conjunction with a K-Nearest Neighbor classifier. Recall the formula for Euclidean distance between two points $\mathbf{p}$ and $\mathbf{q}$ in N dimensions:

$$D(\mathbf{p},\mathbf{q}) = \sqrt{(p_1-q_1)^2+(p_2-q_2)^2+...+(p_N-q_N)^2}.$$

   Also, recall from lecture that we can change the distance function into a weighted Euclidean distance function by applying a different "weight" to each term of the sum under the square root sign:

$$D_{\texttt{weighted}}(\mathbf{p},\mathbf{q}) = \sqrt{w_1(p_1-q_1)^2+w_2(p_2-q_2)^2+...+w_N(p_N-q_N)^2}.$$

   (a) Write a function called **knnWeighted** based on the provided knnStandard function. It should take the same arguments as knnStandard, as well as an additional parameter (sigma). The knnWeighted function should classify each test image (both threes and fives) using a 128x128 size Gaussian distribution as a weighting scheme across the image. In other words, you should weight the pixels near the center of the image highest, and the pixels near the edges of the image lowest. You may use the MATLAB function 'fspecial' to generate the appropriate Gaussian (use "help fspecial" for more details). Once you've generated a 128x128 Gaussian called "gaussian", you can plot it using "imagesc(gaussian)" to see how a particular sigma affects the distribution. You may copy and adapt knnStandard, but be sure to make the appropriate changes so that knnWeighted uses the weighting scheme described above.

(b) Using your new **knnWeighted** function, explore how various values of sigma change the accuracy. Find a value for sigma that produces a higher accuracy than the knnStandard method. You should be able to find a sigma that produces an accuracy of 0.93 or higher. Report the sigma and accuracy you found. In one or two sentences, explain why the Gaussian generated using this sigma produced a higher accuracy in classifiying the test image than the standard (unweighted) K-NN method.

If you'd like to plot your accuracies, you can use the "plot()" command in MATLAB. For example, if you have a vector $y = [5, 2, 4, 3]$, you can plot these values by executing "plot(y)".

3. **Mutual Information** In this part, we will use mutual information to determine which pixel features to pick that provide the most useful information about the image, as described in class.

First, we will write a number of small functions to help us extract the single pixel location with the highest mutual information.

(a) Write a function called **computeLikelihoods** which estimates the likelihoods for **every** pixel in each class ("threes" and "fives"), using the training data. That is, it should return **two** 128x128x2 matrices of likelihoods. In each likelihoods matrix, the first 128x128 layer should contain the likelihoods for when the pixels are off (they have a value of 0.5), and the second layer should contain the likelihoods for when the pixels are on (they have a value of 255.5) (e.g. $P(white|A)$ and $P(black|A)$). You may base this on the likeFromTraining example function covered in class.

(b) Write a function called **computeJoints** which computes the joint probabilities for each pixel and each class using the likelihoods you computed and the **priors** $[0.5, 0.5]$. That is, it should return **two** 128x128x2 matrices of joint probabilities. In each joint probability matrix, the first 128x128 layer should contain the joint probabilities of the pixels being off and the class, and the second layer should contain the joint probabilities of the pixels being on and the class (e.g. $P(black, A)$ and $P(white, B)$).

(c) Write a function called **computeMarginals** which computes the marginal probabilities for each pixel across the entire training set, using the joint probabilities you computed. It should return a 128x128x2 matrix of marginal probabilities, where the first 128x128 layer should contain the marginal probabilities for when the pixels are off, and the second layer should contain the marginal probabilities for when the pixels are on.

(d) Write a function called **computeMutInfo** which uses the previous functions to compute the mutual information at each pixel with each class. It should return a 128x128 matrix. You will be using this to extract the single pixel location with the highest mutual information. You can use imagesc() to visualize the mutual information at each pixel location. Determine the location of the maximum mutual information and report it here, along with the value of the mutual information at that location. Recall that in some cases, you may have a value like $0 * log(0)$, which is 0 when using limits, although MATLAB will compute as NaN ("not a number"). You can get around this by checking if the joint or marginal probability at a given pixel location is 0 and substituting in 0 in your summation, instead of $0 * log(0)$. You can also replace all 0's in your joint and marginal probabilities with a suitably small value (e.g. 10^{-12}), which will not have a noticeable effect on the outcome.

(e) Use this pixel location as input to the provided MAP classifier (classifyTestData) and report the accuracy. See the comments in classifyTestData for more details, or run "help classifyTestData".

Next, we will do the same to extract the second pixel location with the highest mutual information, **given the first location**:

$$I(X_2; C|X_1).$$

This will maximize the information gained by including the mutual information of the second pixel location. To do this, instead of computing the conditional information defined above directly, we will compute

$$I(X_2; C|X_1) = I(X_1, X_2; C) - I(X_1; C).$$

The right side of the equation is computed by first computing the joint mutual information between the pair (X_1, X_2) and the class label C:

$$I(X_1, X_2; C)$$

and then subtracting off the single pixel mutual information

$$I(X_1; C).$$

Here are detailed steps about how to do this.

(a) Make a copy your computeLikelihoods function called **computeLikelihoods2**. This function should take an **additional** argument called "pixel" which will be a 2x1 vector containing the location of the pixel you found in the previous part. It should return two matrices of likelihoods of size 128x128x4. For each matrix, the first layer should contain the likelihoods for when both the provided ("first") pixel **and** the query pixel is off (the "0,0" case from lecture), the second layer should contain the likelihoods for when the first pixel is off and the second pixel is on ("0,1"), the third layer should contain the likelihoods for when the first pixel is on and the second pixel is off ("1,0"), and the fourth layer should contain the likelihoods for when the first **and** second pixels are both on ("1,1"). You should again have one 128x128x4 matrix for each class ("threes" and "fives").

(b) Check your computeJoints function to determine if it will correctly compute the joint probabilities using the likelihoods computed by computeLikelihoods2. If not, you may either rewrite computeJoints to handle both types of likelihoods, or make a copy called computeJoints2 that will handle the likelihoods computed by computeLikelihoods2. It should be capable of accepting two 128x128x4 matrices of likelihoods as arguments and return two 128x128x4 matrices of joint probabilities (one for each class).

(c) Similarly, check your computeMarginals function to determine if it will correctly compute the marginal probabilities using the joint probabilities computed above.

(d) Finally, make a copy of your computeMutInfo called **computeMutInfo2** that computes the joint mutual information between the pair of pixels (X_1, X_2) and the class C, where X_1 is the most informative pixel from the first step and X_2 is a candidate second pixel. Just as in the previous section, use the output from the previous functions to compute these mutual informations. This should return a single 128x128 matrix of joint mutual informations. **Use the same trick as before to deal with cases like $0 * log(0)$.**

(e) **Information Gain** Now that we have the mutual informations of single pixels $I(X_1; C)$, and the mutual information of single pixels **paired with the first pixel** $I(X_1, X_2; C)$, we can compute the **information gain**:

$$I(X_2; C|X_1) = I(X_1, X_1; C) - I(X_1; C).$$

Find the pixel location that maximizes the information gain given the first pixel. Report the location and **information gain** at the location.

You can visualize the mutual information and information gain (or any "layer" of the output you have produced) by using imagesc(). In particular, generate images of

- The image of mutual informations for one pixel.
- The image of joint mutual informations for the best pixel from the first step paired with each possible second pixel.
- The image of the information gain from the second pixel given the first pixel. This should just be the difference of the first two images.

What to turn in

- Every matlab function you were asked to create. If a function can handle both types of input (e.g. computeJoints.m or computeMarginals.m), note it as a comment in the function.
- A single PDF containing all your code, and the values you are asked to report, and the 3 images you produced in part 3. It should be named lastnameFirstname_assign3.pdf.

Email these files as a zip named lastnameFirstname_assign3.zip to the TA.