Review for Exam 1

Erik G. Learned-Miller
Department of Computer Science
University of Massachusetts, Amherst
Amherst, MA 01003

March 7, 2011

Abstract

This document reviews material you will need for Exam 1. I recommend you do each problem here. If you can do them all without referring to other material, you should be in good shape for the exam. I recommend

that you review your notes and the class documents before trying to do the material so that you know where your deficits are. .

1 Basic notation

You should understand the following types of notation.

- Summation ($\sum$). Example:

$$\sum_{i=1}^{N} X_i = X_1 + X_2 + ... + X_N. \tag{1}$$

- Product ($\prod$). Example:

$$\prod_{i=1}^{N} X_i = X_1 \times X_2 \times ... \times X_N. \tag{2}$$

- When we perform optimization of a function with respect to some changeable parameter, we are usually interested in either the minimum or maximium value of the function itself, or, alternatively, we are interested in the values of the parameter that achieved the maximum or minimum.

  Suppose we are trying to find the maximum value of a function $f(x, y)$ when y is set to the value 3 and x can be any value from some set S. To specify the maximum value of the function we write:

$$\max_{x \in S} \; f(x, y = 3). \tag{3}$$

  We frequently assign this maximum to some variable, as in:

$$F^* = \max_{x \in S} \; f(x, y = 3). \tag{4}$$

- If we are interested in the value of x which gave us this optimum, we use the $\arg\max$ notation:

$$\arg\max_{x \in S} \; f(x, y = 3). \tag{5}$$

  In this case, of course, the maximizing x might be called x^*:

$$x^* = \arg\max_{x \in S} \; f(x, y = 3). \tag{6}$$

2 Basic probability

- Know all the material from the *Review of Basic Probability* handout.
- Particularly focus on marginalization and Bayes rule.
- Suppose we are trying to classify images into one of several classes c_1, c_2, and c_3, and to do it we are using feature values f_1 and f_2. Know what is meant by the *likelihoods of the classes*, the *priors of the classes*, and the *posterior probabilty* for each class.

- Marginalization: If you are given a probability distribution for each pair of possible values of $Prob(X, Y)$, then you should be able to computer $Prob(X)$ and $Prob(Y)$ for each possible value of X and Y.
- Know the definition of statistical independence: For two random variables X and Y, X and Y are statistically independent if and only if $P(X|Y) = P(X)$ for all X and Y. Or alternatively, $P(X, Y) = P(X)P(Y)$. Be able to derive one of these formulas from the other (multiply both sides by $P(Y)$.
- Conditional probability. Be able to compute $P(X|Y)$ from $P(X, Y)$ and $P(Y)$.
- Be able to given reasonable estimates of $P(X)$, $P(X|Y)$, and $P(X, Y)$ from raw data.

3 Basic Matlab familiarity

- Know how to turn the values in a matlab matrix into a vector (use the colon operator).
- Know how to transpose a matrix and what this means (you can use either the quote operator (') or use the transpose command. This can used to turn a row vector into a column vector or vice versa.
- Understand the structure of a "grayscale" or "scalar-valued" image. It is simply a 2-dimensional array of numbers.
- Understand the structure of a "color" image: it is a 3-dimensional array of numbers in which the first "layer" represents red, the second layer represents green, and the third represents blue.
- IMPORTANT: Understand that a *grayscale* or *scalar-valued* image can be rendered in color using a *look-up-table*, which is essentially a scheme for doing color-by-number. That is, for each value in the image, the computer looks up the red-green-blue (RGB) color that should be used for that particular number.
- Know the repmat command in matlab. Know that it can be used to avoid doing for loops, and that for loops are slow in matlab.

4 Image comparison

- Understand the "sum of squared differences" between two scalar-valued images I and J:

$$SSD(I, J) = \sum_{i=1}^{\#of pixels} (I_i - J_i)^2.$$

This is the same thing as the "Euclidean distance" except that the Eucildean distance has a square root:

$$\begin{aligned} D_{Euclid}(I,J) &= \sqrt{\sum_{i=1}^{\#of pixels} (I_i - J_i)^2} \\ &= \left(\sum_{i=1}^{\#of pixels} (I_i - J_i)^2 \right)^{\frac{1}{2}}. \end{aligned}$$

- Understand the "sum of absolute differences":

$$\begin{aligned} D_{abs}(I,J) &= \sum_{i=1}^{\#of pixels} |I_i - J_i| \\ &= \left(\sum_{i=1}^{\#of pixels} |I_i - J_i| \right)^{\frac{1}{1}}. \end{aligned}$$

  Understand why one might want to use one instead of the other (sum of squared differences weighs larger errors more heavily, on average).

- Know the general formula for an L_p distance:

$$L_p(I,J) = \left(\sum_{i}^{\#of pixels} (I_i - J_i)^p \right)^{\frac{1}{p}}. \tag{7}$$

  Note that the Euclidean distance is the same as the L_2 distance ($p = 2$) and that the sum of absolute differences is the same as the L_1 distance ($p = 1$).

5 Transforms

You should understand the properties of the following sets of transformations.

- **Shifting only.** Also called **Translations**. Preserves orientation and all the properties of rigid transformations.
- **Rigid. (translation + rotation)** Preserves areas (or volumes if in 3-D) and all the properties of similarity transforms.
- **Similarity (rigid + scale)**. Preserves angles, straightness of lines, paralllelness of lines.
- **Linear transformations.** This is the family of transformations that is represented by ALL full rank 2x2 matrices. (Full rank just means the matrix doesn't squash the object to have no width or no height. For

example, a matrix filled with zeroes would transform all coordinates to (0,0), so the object would effectively be shrunken to have 0 size.) Linear transformations preserve the parallelness of lines and the straightness of lines. These transformations include, as special cases, rotations, scalings, shearings, and flips. *IMPORTANT:* Note that linear transformations do NOT include translations!

- **Affine transformations (Linear + translations).** These transformations are exactly like the linear transformations, except that they also include the translations. This allows you to, say, both rotate and move an object, instead of just rotating it. This is important since linear transformations always transform with respect to the origin (0,0). So if you want to rotate something about its center, and the center is not at the origin, then you will need affine transformations.

5.1 Transforming images pixel by pixel

You should understand the programs we wrote in class to rotate images. These are called **rotate1.m** and **rotate2.m** on the class web page. In particular, you should understand why rotate2.m was necessary to "fill the holes" created by rotate1.m.

6 Classification

- Understand Bayes rule. Practice it many times so you are fluid with it. Try making up some tables of $p(x|c)$ where x is a feature value and c is a class, and then try to compute $p(c|x)$. You will also need, of course, $p(c)$ the prior probabilty of a class. *If you don't understand this, you better get help before the test.*
- Know what the Bayes Error rate is. Given the true likelihoods (not the ones estimated from data) for each class, and the true priors (not the ones estimated from data), the Bayes error rate is the lowest possible error rate fthat any classifier can achieve.
- Given the true likelihoods and the true priors, using Bayes rule to compute the posteriors, and picking the class with the highest posterior probability gives a Bayes Optimal Classifier, which is a classifier that achieves the Bayes error, i.e. the minimum possible error.

7 Understanding the sizes of sets

- How many distinct 10x10 binary images are there? Answer: 2^{100}.
- How many 1000x1000 color images are there if each color channel (red, green, blue) can take on 256 colors? First, calculate the number of colors

per pixel:

$$\begin{aligned} c &= 256 * 256 * 256 && (8)\\ &= 2^8 * 2^8 * 2^8 && (9)\\ &= 2^{24} && (10)\\ &= 2^{20} * 2^4 && (11)\\ &\approx 1,000,000 * 16 && (12)\\ &= 16\ million. && (13) \end{aligned}$$

Given about 16 million colors per pixel, the number of 1000x1000 images is

$$(16\ million^{1000*1000}) = (16\ million)^{million}.$$

According to some estimates, there are about 2^{80} particles in the known universe.

- Given 100 x translations of an image, 100 y translations, 1000 rotations, and 500 different scales, how many different images tranformations could I produce? I'll let you do this one yourself.