Winter 2017: EECS 545 Homework 3

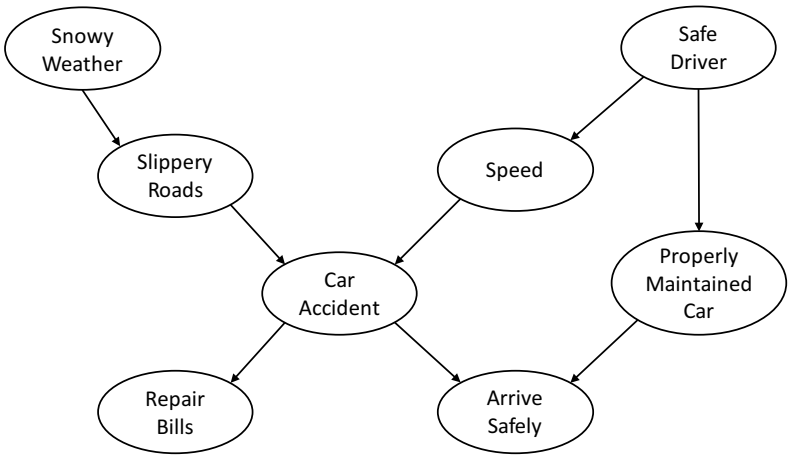
Due: 14 April 2017, 11:59 PM EST

Homework Policy: Working in groups is fine, but each member must submit their own writeup. Please write the members of your group on your solutions. There is no strict limit to the size of the group but we may find it a bit suspicious if there are more than 4 to a team. Questions labelled with **Challenge** are not strictly required, but you'll get some participation credit if you have something interesting to add, even if it's only a partial answer. For coding problems, please report your results (values, plots, etc.) in your written solution, and append the code in the end.

Late submission will NOT be accepted.

1 Properties of Graphical Models

(10 points) In this question, you will explore the independence properties of Bayesian networks. For each question, provide your answer together with a brief explanation. Consider the following Bayes net:



(1) (2 points) Is arriving safely independent of snowy weather?

(2) (2 points) Is a properly maintained car independent of a repair bill?

(3) (2 points) Is a properly maintained car conditionally independent of snow weather given a repair and a safe driver?

(4) (2 points) Is a repair bill conditionally independent of arriving safely, given a car accident?

(5) (2 points) Is it possible to make properly maintained car independent of slippery roads given a repair bill by reversing one edge in the network? If so, which edge?

2 K-medoids

(5 points) Let $\{\mathbf{x}_i\}_{i=1}^n$ be data points. The K-medoids algorithm is a clustering algorithm similar to K-means. Let $d(\mathbf{x}, \mathbf{x}')$ denote the distance between $\mathbf{x}$ and $\mathbf{x}'$, and let $m_1, \cdots, m_k$ denote cluster centroids. K-means iterates

1. $C(i) = \text{argmin}_{\{l=1,\cdots,K\}} d(\mathbf{x}_i, m_l)$

2. m_l = sample mean of data points $\mathbf{x}_i$ where $C(i) = l$

In K-medoids, the second step is replaced with

$$m_l = \underset{\{m \in \mathbf{x}_i : C(i)=l\}}{\operatorname{argmin}} \sum_{j:C(j)=l} d(\mathbf{x}_j, m)$$

Note that in K-medoids, the cluster center must be a member of the cluster.

(1) (2 points) Argue that K-medoids is more robust to outliers than K-means.

(2) (3 points) Other than robustness, what is a significant advantage of K-medoids to K-means?

3 Expectation Maximization (E-M) for Naive Bayes

(20 points) In Homework 2, we implemented a naive Bayes classifier. In reality, it is hard to obtain many labeled data (because creating labels need manual effort) but relatively easy to obtain vast amounts of unlabeled data. How can we leverage the information in the unlabeled data in training a naive Bayes classifier? It turns out we can use the E-M algorithm.

Suppose we have l labeled data points $\left\{\left(\mathbf{x}^{(1)}, y^{(1)}\right), \cdots, \left(\mathbf{x}^{(l)}, y^{(l)}\right)\right\}$ and u unlabeled data points $\left\{\left(\mathbf{x}^{(l+1)}\right), \cdots, \left(\mathbf{x}^{(l+u)}\right)\right\}$, where the input $\mathbf{x}$ has dimension m. We have the following additional assumptions (note they are different from those in Homework 2):

- The data is binary-valued, i.e. each coordinate x_j takes a value of 0 or 1 for all $j \in \{1, \cdots, m\}$.
- The class label $y \in \{1, \cdots, k\}$.
- The distribution of the data follows from the naive Bayes assumption:

$$P(\mathbf{x}, y) = P(y) \prod_{j=1}^{m} P(x_j | y) \tag{1}$$

$$P(y = c) = \pi_c, \ c = 1, \cdots, k \tag{2}$$

$$P(x_j = 1 | y = c) = \theta_{jc}, \ d = 1, \cdots, m, c = 1, \cdots, k \ . \tag{3}$$

Here, π_c are multinomial probabilities (i.e. $\sum_{c=1}^{k} \pi_c = 1$). θ_{jc} are Bernoulli probabilities. We can express the joint distribution in a one-line formula (instead of multiple if-else statements) as:

$$P(\mathbf{x}, y) = \prod_{c=1}^{k} \left(\pi_c \prod_{j=1}^{m} \theta_{jc}^{x_j} (1 - \theta_{jc})^{1 - x_j} \right)^{\mathbf{1}_{\{y=c\}}} , \tag{4}$$

where $\mathbf{1}_{\{z\}}$ is the indicator function: $\mathbf{1}_{\{z\}} = 1$ if z is true; 0 otherwise. Since we have unlabeled data, our goal is to maximize the following hybrid log-likelihood of labeled and unlabeled data:

$$\sum_{i=1}^{l} \log P(\mathbf{x}^{(i)}, y^{(i)}) + \lambda \sum_{i=l+1}^{l+u} \log P(\mathbf{x}^{(i)}) \ . \tag{5}$$

$\lambda > 0$ is a predefined constant that balances the two log-likelihood terms. Now let's walk through the derivation of E-step and M-step.

(1) (5 points) In the E-step, we want to make a guess on the missing labels of unlabeled data. In particular, our guess is probabilistic (a "soft guess"): each unlabeled data point takes label c with posterior probability $q_c^{(i)} = P(y^{(i)} = c | \mathbf{x}^{(i)})$ for $i = l+1, \cdots, l+u$. Express $q_c^{(i)}$ in terms of current model parameters π_c, θ_{jc} and the feature values $x_j^{(i)}$, for $j = 1, \cdots, m$, $c = 1, \cdots, k$.

(2) (3 points) In the M-step, we want to re-estimate the model parameters π_c and θ_{jc} based on the guessed labels from the E-step. First fill out the missing term in the following complete-data log-likelihood function $Q(\boldsymbol{\pi}, \boldsymbol{\theta})$:

$$Q(\boldsymbol{\pi}, \boldsymbol{\theta}) = \sum_{i=1}^{l} \log P(\mathbf{x}^{(i)}, y^{(i)}) + \lambda \cdots \tag{6}$$

[**Hint**: the term should contain posterior probabilities $q_c^{(i)}$ and complete-data probabilities $P(\mathbf{x}^{(i)}, c)$ for $i = l+1, \cdots, l+u$ and $c = 1, \cdots, k$.]

(3) (12 points) Derive the M-step for this problem. That is, we want to update model parameters π_c and θ_{jc} such that they maximize the current $Q(\boldsymbol{\pi}, \boldsymbol{\theta})$. Express π_c and θ_{jc} in terms of

- feature values $x_j^{(i)}$ of both labeled and unlabeled data;
- known labels $y^{(i)}$ of labeled data;
- guessed label probabilities $q_c^{(i)}$ of unlabeled data, from the E-step;
- known constants l, u, λ.

[**Hint:** this is a constrained optimization problem with the equality constraint $\sum_{c=1}^{k} \pi_c = 1$.]

4 K-means and Gaussian mixtures for image compression

(40 points) In this problem, we will apply the K-means algorithm to lossy image compression, by reducing the number of colors used in an image. Canvas contains a 512×512 image of a mandrill represented in 24-bit color. This means that, for each of the 262,144 pixels in the image, there are three 8-bit numbers (each ranging from 0 to 255) that represent the red, green, and blue intensity values for that pixel. The straightforward representation of this image therefore takes about $262144 \times 3 = 786432$ bytes (a byte being 8 bits). To compress the image, we will use K-means to reduce the image to $K = 16$ colors. More specifically, each pixel in the image is considered a point in the three-dimensional (r, g, b) color-space. To compress the image, we will cluster these points in color-space into 16 clusters, and replace each pixel with the closest cluster center. Follow the instructions below. Be warned that some of these operations can take a while (several minutes even on a fast computer!)

Note: This problem consists of implementing two algorithms: K-means and GMM. The K-means part will be 20 points, and the GMM part will be 20 points.

(1) Preparation: read and write image files. We try to provide pointers for Matlab, Python, and R users. For users of other languages, please find appropriate packages to read and write images in TIFF format, or you can learn to use Matlab, Python, or R.

First, download `mandrill-large.tiff` and `mandrill-small.tiff` from Canvas.

- For Matlab users: to read the image, use

```
A = double(imread('mandrill-large.tiff'));
```

Now `A` is a "three-dimensional array", and `A(:,:,1)`, `A(:,:,2)`, `A(:,:,3)`, are 512 × 512 matrices that respectively contain the red, green, and blue values for each pixel. To save the image to file, use

```
imwrite(uint8(round(A)), '<filename>.tiff');
```

- For Python users, install the `OpenCV` package (using `conda` might be an easy way[1]; installation command might be different for Python 2 and 3). Import the package using `import cv2`. To read the image, use

```
A = cv2.imread('mandrill-large.tiff').astype('float64')
```

Now `A` is a "three-dimensional array", and `A[:,:,0]`, `A[:,:,1]`, `[:,:,2]` are 512 × 512 matrices that respectively contain the blue, green, and red[2] values for each pixel. To save the image to file, import Numpy using `import numpy as np`, then use

```
cv2.imwrite('<filename>.tiff', np.rint(A).astype('uint8'))
```

[1] `http://stackoverflow.com/questions/23119413/how-to-install-python-opencv-through-conda`

[2] The order is different from red, green, blue, but it shouldn't matter as long as you use the original order in your code.

- For R users, install packages `pixmap` and `rtiff` (it should be straightforward using the 'R package installer' in R-studio). Import the packages using `library(pixmap);library(rtiff)`. To read the image, use

```
A <- readTiff("mandrill-large.tiff", pixmap=FALSE)
```

Now `A$r`, `A$g`, and `A$b` are 512×512 matrices that respectively contain the red, green, and blue values for each pixel. To save the image to file, use

```
writeTiff(newPixmapRGB(A$r, A$g, A$b), '<filename>.tiff')
```

Before manipulating these matrices in your code, remember to convert their elements from `uint8` into `numeric`: `A$r <- apply(A$r, 2, as.numeric)`; similarly for `A$g` and `A$b`.

(2) Since the large image has 262144 pixels and would take a while to cluster, we will instead run vector quantization on a smaller image. Repeat part (1) with `mandrill-small.tiff`. Treating each pixel's (r, g, b) values as an element of $\mathbb{R}^3$, implement your own K-means algorithm with $K = 16$ clusters on the pixel data from this smaller image, iterating (preferably) to convergence, but in no case for less than 30 iterations. For initialization, set each cluster center to the (r, g, b)-values of a randomly chosen pixel in the image.

(3) Take the matrix `A` from `mandrill-large.tiff`, and replace each pixel's (r, g, b) values with the values of the closest cluster center. Write out the new image, and compare it visually to the original image. Submit all your code and a printout of your compressed image.

(4) If we represent the image with these reduced (16) colors, by (approximately) what factor have we compressed the image?

(5) Repeat (2), (3), and (4) by implementing your own Gaussian mixtures (with full covariances) with $K = 5$; for (2) and (3), use the MAP estimation for the latent cluster assignment variable for each pixel. In addition, after training the GMM, report the model parameters $(\mu_k, \Sigma_k) : k = 1, ..., 5$.

5 Principal Component Analysis

(15 points) This problem will help you gain intuition about the solution of PCA. Suppose $\mathbf{x}$ is a vector of n random variables. Let $var(\mathbf{x}) = \Sigma \in \mathbb{R}^{n \times n}$ be known. Suppose $c \in \mathbb{R}^n$ is a vector of constants. We seek the c that maximizes $var(c^T\mathbf{x}) = c^T\Sigma c$. Because one possible way of achieving maximum variance is to simply pick an arbitrarily large c, we constrain it to have unit length. Pose this as a constrained optimization problem. What is the solution for c? How can we make sure that it is unique? [**Hint**: Use the Eigen decomposition of Σ]