6.047 / 6.878 Computational Biology: Genomes, Networks, Evolution
Fall 2008

6.047/6.878 Fall 2008 - Problem Set 2

Due: September 24, 2008 at 8pm

1. **Bayes Rule and Naive Bayes Classifier.** In this problem we will familiarize ourselves with Bayes Rule and the use of Naive Bayes for classification. Assume that the probability of there being rain on any given day is 0.1, and that the probability of getting in a car accident is 0.05.

   (a) Assume these two probabilities are independent. What is the probability of there being a day that is both rainy and you get in a car accident? What is the conditional probability of getting into a car accident today, given that you already know today is rainy?

   (b) From vehicle accident records, assume that we've determined that the probability of it being a rainy day, given that a car accident was observed is 0.4. Using Bayes' Rule and the prior probabilities, what is the conditional probability of getting into a car accident today, given that you already know today is rainy?

   (c) Why are the conditional probabilities you computed in parts (a) and (b) different?

   (d) The following table describes features for DNA sequences and the class of corresponding DNA sequence. GC content describes the fraction of Gs and Cs (as opposed to As and Ts) in the DNA sequence. Complexity describes the degree of randomness of the sequence. Repeats are stretches of DNA that are highly repetitive and occur multiple times in a genome. Motifs are sites where transcription factors bind.

GC Content	Length	Complexity	Class
Low	Long	High	Gene
Low	Long	Low	Gene
High	Long	High	Repeat
Medium	Short	High	Motif
Medium	Short	Low	Motif
High	Long	Low	Repeat
High	Short	High	Motif
Medium	Long	High	Gene
High	Long	Low	Repeat
High	Short	High	Motif

   Use the Naive Bayes classifier to predict this genes class (**show your work**):

GC Content	Length	Complexity	Class
Medium	Long	Low	?

2. **Bayesian Decision Theory.**

   For many classification problems, our objective will be more complex than simply trying to minimize the number of misclassifications. Frequently different mistakes carry different costs. For example, if we are trying to classify a patient as having cancer or not, it can be argued that it is far more harmful to misclassify a patient as being healthy if they have cancer than to misclassify a patient as having cancer if they are healthy.

   In the first case, the patient will not be treated and would be more likely to die, whereas the second mistake involves emotional grief but no greater chance of loss of life. To formalize such issues, we introduce a *loss function* L_{kj}. This function assigns a loss to the misclassification of an object as class j when the true class is class k. For instance, in the case of cancer classification L_{kj} might look like (where the true class k is along the x axis):

	True Cancer	True Normal
Predict "Cancer"	0	1
Predict "Normal"	1000	0

Which says that there is a loss of 1 if we misdiagnose a healthy patient as having cancer, but a much bigger loss of 1000 if misdiagnose a patient with cancer as normal.

(a) In classification, we do not know the actual class k of a data sample d (a patient in our example). Thus, when computing the loss of predicting class j for sample d, we can only calculate *expected* loss. Knowing the class priors $P(k)$ and likelihoods $P(d|k)$, derive the equation for the expected loss $E_k[L]$ of assigning sample d to class j.

(b) Say, for every data point we wish to choose the class which minimizes the quantity you derived in part (a). For the case where the loss matrix is $L_{kj} = 1 - I_{kj}$ (I_{kj} are the elements of the identity matrix), determine what such a decision rule would effectively do.

(c) What is the interpretation of the loss matrix in part (b).

3. **K-means clustering**. In this problem you will implement the K-means clustering algorithm.

(a) Implement K-means using any programming language. Include the source code with your write-up. At a minimum, your implementation should:

i. Take a tab-delimited file with an $N \times M$ matrix of N data points with M features each. In other words, each row is a data point and each column is a feature value associated with that data point.

ii. Accept K (the number of clusters) as a parameter.

iii. Output the M-dimensional mean vectors for each of the K clusters, and also an assignment for each clustered data point.

iv. Your stopping/convergence condition can be as simple or complex as you wish.

Note: Use `generate_clusters.py` *to make testing input data. You will find that the easiest data to cluster will be spherical (unit variance in both dimensions) and well separated (cluster means farther than 2 standard deviations apart). Run the script with no arguments to display its help information. While* `generate_clusters.py` *will create only two dimensional data points, your algorithm should accept an arbitrary number of dimensions (e.g. 22-dimensions in part (d)).*

(b) Implement the fuzzy K-means algorithm with the same features listed above. Include source code in your write-up.

(c) We will first apply both of these algorithms to synthetic data. Use the provided Python script `generate_clusters.py` for this purpose.

i. Generate $N = 50$ points from 3 Gaussians (150 points total). Generate a data set that is difficult to cluster and requires a nontrivial solution. For example, means should be within 2 standard deviations of each other and one of the Gaussians should be elongated, not spherical.

- Run both algorithms on this data with $K = 2, 3$ and 4 cluster centers. Discuss what happens when the number of cluster centers is not 3. Include the parameters chosen for data generation.
- To visually assess the accuracy of your algorithms for the runs above, include at least one clustering plot for each algorithm. The clustering plot should show a scatter of the data points and indicate BOTH the "correct" and "predicted" cluster for each point. For example, each point might be drawn as a *, $+$, or - according to the correct cluster, and colored red, green, or blue according to your predicted cluster. (Note that the "correct" cluster is provided as a third column in the output of `generate_clusters.py`.) Optionally, you may want to also plot the final predicted cluster centroids. (You do NOT need to include any code for generating plots, just the plots themselves)

ii. The K-means algorithm tries to find the cluster centers μ that minimize the objective

$$\sum_{i=1}^{K} \sum_{x_j \in C_i} ||x_j - \mu_i||_2^2,$$

where C_i is the set of points assigned to cluster i. K-means is a greedy algorithm and is not guaranteed to find the global minimum of this objective. In fact, the quality of the resulting solution often depends significantly on the cluster initializations. For $K = 3$, repeat K-means 100 times with different random cluster initializations, and report the best μ values (i.e., which minimize the objective). How much variation was there?

(d) Now we will apply your K-means implementation to a real-world problem, described in the paper *Boshoff et al. (2004) "The Transcriptional Responses of Mycobacterium tuberculosis to Inhibitors of Metabolism." Journal of Biological Chemistry. 279(38):40174-40184.*

We provide you with a subset of their data (see 'description.txt' file). Each of the 3 data files represents the treatment of the TB bacteria, *Mycobacterium tuberculosis*, by a different class of drugs. One class inhibits cell wall synthesis, one blocks translation, and the last blocks iron uptake. For each of the 3924 genes the authors measured the gene expression for specific drugs within each class. We are interested in clustering the genes to find groups of genes that are coordinately regulated in their response to each drug. This may happen, for example, if there is one gene which acts as a "master switch", regulating various other downstream genes.

i. For each of the 3 classes of drugs, use one of your programs to cluster the 3924 genes into $K = 150$ clusters. Feel free to use a different K if you find it produces better results. Since plotting high dimensional data is difficult, report some other statistics about your clusters (i.e. such as a bar chart of their sizes). *Note: since this is real data, do not worry if your clusters are not as clearly defined. There is no known answer.*

ii. Each gene is represented as a 18-22 length vector, called *an expression profile*, which contains the data: log(expression in drugged sample/expression in control sample) for 18-22 different conditions. The length (ℓ_2 norm) of each expression vector indicates how much the drug changes the gene's expression relative to the control (no drug present). Find the gene that is most sensitive for each drug (what is it?), and then look at what cluster it belongs to. Do a Google search for some of the genes in each cluster (see 'rowNames.txt' file). List a few functions that these genes are associated with.

iii. For extra credit, plot the results of your clustering in such a way as to visualize your cluster centers, cluster assignments, and actual cluster labels for these data sets.

iv. For more extra credit, see what commonalities you can find between the genes in each cluster (*why* might they be co-regulated?). Are there conserved promotor motifs? Do they belong to the same operon? Perhaps genes in expression clusters are more likely to be neighbors on the chromosome than genes chosen at random, since genes in operons are cotranscribed. Note: this is exploratory work – we do not know the answers!

4. **Grad only problem: Expectation Maximization in Gaussian Mixtures**. Recall from lecture that fuzzy K-means is a form of expectation maximization using a mixture of Gaussians. We are going to show that in this problem.

For this problem, we are going to assume that we are given an independent, identically distributed sample of N points $x_1, \ldots, x_N$. We are told that each x_i is drawn from one of K Gaussian distributions with unit variance but with unknown means μ_k. Consequently each x_i can be associated with a label Y_i from the set $\{1, \ldots, K\}$. But we are not told what these labels are. The Gaussians are equally probable, i.e., the prior probability that a data point is drawn from one of the Gaussians is $\frac{1}{K}$. Our goal is to estimate (1) the means μ_k for each of the K Gaussians, and (2) the probability that each x_i belongs to each Gaussian.

To do so, we will use Expectation-Maximization (EM). As you learned in lecture, EM is a re-estimation procedure that – loosely speaking – iterates between estimating the missing labels (Y_i in our case) given the current best parameter values (μ_k in our case) – this is the E-step – and then re-estimating the parameters given these estimates by choosing these parameters to maximize the expected log-likelihood of the data and the labels – this is the M-step.

(a) Suppose that X_i is sampled from a Gaussian distribution with unit variance: $X_i \sim \mathcal{N}(\mu_k, 1)$. Write out the equation for $\Pr(X_i = x_i; \mu_k)$.

Note: The notation "; μ_k" just clarifies that the probability distribution for X_i uses the parameter μ_k for the Gaussian's mean.

(b) Now lets estimate the probability of the unobserved labels. Given a set of K means $\mu = \{\mu_1, \ldots, \mu_K\}$ and a data point x_i, write the equation for $\Pr(Y_i = k \mid X_i = x_i; \mu)$, using the result from part (a).

Hint: Use Bayes Rule.

(c) Assume for a moment we knew the labels associated with each point. Write out the equation for $\log \Pr(x_1, \ldots, x_N, y_1, \ldots, y_N; \mu)$, the log-likelihood of the N points and their labels. We denote the log-likelihood as $L(\mu)$, making clear its dependence on the parameters μ. Notice that $L(\mu)$ is a function of $x_1, \ldots, x_N, y_1, \ldots, y_N$.

(d) However, we do not know the labels. But part (b) did provide us with $\Pr(Y_i = k \mid X_i = x_i; \mu)$. We can use this to find the expected log-likelihood, $Q(\mu)$. The expected value of a general function $f(x)$ is just $\sum_x \Pr(x) f(x)$. Using your answer from (b) and (c), write out the equation for the expected value of the log-likelihood, $Q(\mu) = E[L(\mu)]$, where the expectation is of the Y variables, using the conditional the probability distribution $\Pr(Y_i = k \mid X_i = x_i; \mu^{(t)})$. The notation $\mu^{(t)}$ refers to the current best estimate of the parameters. You should consider $\mu^{(t)}$ a constant quantity, provided to you (from the previous iteration). The quantities computed here are the output of the **E-step** of EM. **Show your derivation.**

Note: Feel free to replace constants that will not influence subsequent parts of this problem with a symbol (e.g. C).

Hint: You may want to use the linearity of expectation.

(e) During the **M-step** of EM, we wish to choose $\mu = \{\mu_1, \ldots, \mu_K\}$ to maximize the expected log-likelihood (which, notice, is a function of μ) generated during the previous E-step. To do so, we need to calculate the derivatives $\frac{\partial Q(\mu)}{\partial \mu_k}$. Derive the equation for these derivatives.

(f) To find the μ which maximize the expected log-likelihood, take your answer from part (e), set the derivative $\frac{\partial Q(\mu)}{\partial \mu_k} = 0$, and solve for μ_k. We denote this value of μ as $\mu^{(t+1)}$: these would then be used in the next iteration of EM.