1. Suppose you wish to do predictive text entry on a cell phone without a built-in dictionary. In predictive text entry, the observations are numbers and each number is associated with 3 or 4 letters (e.g., 2 → ‘a’, ‘b’, or ‘c’). The goal is to decode the intended word given the corresponding sequence of digits. Formulate this problem as a hidden Markov model, answering the following.

(a) What are the hidden states?
(b) What are the emission probabilities? How could you estimate parameters for the transition probabilities from freely available data on the internet? Describe what data you would use and how.
(c) Suppose that, with probability $p$, a user accidentally presses a random digit instead of the correct one. How can you modify your model to handle this?
(d) What are the limitations of this HMM as a model for predictive text entry?

2. (Grads only.) Consider a discrete domain defined by $n$ random variables, $\{X_1, \ldots, X_n\}$, each with $k$ values. We wish to represent probability distributions over this domain using mixture models in which each mixture component assigns a non-zero probability to exactly one point (atomic event) in the event space. That is, each conditional distribution $P(x|C = c_i)$ has the value 1 for exactly one configuration $x'$ and the value 0 for all others. We will refer to this as a mixture of points (MOP) model.

(a) Describe a procedure (which need not be efficient) for representing an arbitrary Bayesian network as a MOP. How many mixture components are required (in general)?
(b) Given a training set of $m$ fully observed examples (all unique), describe a procedure for constructing a MOP that maximizes the likelihood of the training data. How many mixture components are required?
(c) Suppose some of the values in some of the training examples are missing at random. Now how many mixture components are required to construct a MOP that maximizes the likelihood of the (partially observed) training data? Explain.
(d) Do you think MOP models would be useful for solving real-world problems? Why or why not?
3. **Programming project.** Implement the EM algorithm for mixtures of Gaussians in your choice of programming language. (C, C++, Java, Perl, Python, and OCaml are all fine. Please ask about any others.) Assume that means, covariances, and cluster priors are all unknown. For simplicity, you can assume that covariance matrices are diagonal (i.e., all you need to estimate is the variance of each variable). Initialize the cluster priors to a uniform distribution and the standard deviations to a fixed fraction of the range of each variable. Your algorithm should run until the relative change in the log likelihood of the training data falls below some threshold (e.g., stop when log likelihood improves by < 0.1%).

Your program should read in data files in the following format:

```
<# of examples> <# of features>
<ex.1, feature 1> <ex.1, feature 2> ... <ex.1, feature n>
<ex.2, feature 1> <ex.2, feature 2> ... <ex.2, feature n>
... 
```

And output a model file in the following format:

```
<# of clusters> <# of features>
<clust1.prior> <clust1.mean1> <clust1.mean2> ... <clust1.var1> ... 
<clust2.prior> <clust2.mean1> <clust2.mean2> ... <clust2.var1> ... 
... 
```

Train and evaluate your model on the Wine dataset, available from the course Web page. Each data point represents a wine, with features representing chemical characteristics including alcohol content, color intensity, hue, etc. We provide a single default train/test split, with the class removed. You can find the full dataset and more information in the UCI repository (and linked from the course Web page). Start by using 3 clusters, since the Wine dataset has three different classes. Evaluate your models on the test data. (I recommend reading in the test data to evaluate your models as you learn them, one iteration at a time.)

**NOTE:** Probabilities can easily become too small to store in floats or doubles. To avoid underflow:

- Work with logs of probabilities, not probabilities.
- To compute the log of a sum of exponentials, use the “log-sum-exp” trick:

\[
\log \sum_i \exp(x_i) = x_{\text{max}} + \log \sum_i \exp(x_i - x_{\text{max}})
\]

(Your program will only be evaluated on the Wine dataset, so if you do not encounter underflow there, then you may omit the above techniques.)

Answer the following questions with both numerical results and discussion.

(a) List the log likelihood of the training data and the test data at the start of each of the first 20 iterations. (Note: the test data is only used for evaluating the model being learned on the training data.) Approximately how long does it take to converge?
(b) Run the algorithm 10 times with different random seeds. How much does the number of iterations and final log likelihood change from run to run?

(c) Infer the most likely cluster for each point in the training data. How does the true clustering (see wine-true.data) compare to yours?

(d) List the training and test set log likelihoods, varying the number of clusters from 1 to 10. Discuss how the training set log likelihood varies and why. Discuss how the test set log likelihood varies, how it compares to the training set log likelihood, and why. Finally, comment on how train and test set performance with the “true” number of clusters (3) compares to more and fewer clusters and why.