Massachusetts Institute of Technology

6.867 Machine Learning, Fall 2006

Problem Set 5: Solutions

1. (a) For the LM, there is no need to iterate; the maximum likelihood estimates are easy to derive; they are simply the normalized counts.

The updates for T and D are:

$$T'(f \mid e) = \frac{1}{Z_T(e)} \sum_{\substack{i,j,k \\ \mathbf{e}_i^{(k)}=e,\, \mathbf{f}_j^{(k)}=f}} \frac{D(a_j = i \mid j, \ell, m) T(\mathbf{f}_j^{(k)} \mid \mathbf{e}_i^{(k)})}{\sum_{i'=0}^{\ell} D(a_j = i' \mid j, \ell, m) T(\mathbf{f}_j^{(k)}, \mathbf{e}_{i'}^{(k)})}$$

$$D'(a_j = i \mid j, \ell, m) = \frac{1}{Z_D(j, \ell, m)} \sum_{\substack{k \\ |\mathbf{e}^{(k)}|=\ell,\, |\mathbf{f}^{(k)}|=m}} \frac{D(a_j = i \mid j, \ell, m) T(\mathbf{f}_j^{(k)} \mid \mathbf{e}_i^{(k)})}{\sum_{i'=0}^{\ell} D(a_j = i' \mid j, \ell, m) T(\mathbf{f}_j^{(k)}, \mathbf{e}_{i'}^{(k)})},$$

where Z_T and Z_D are normalization constants. One can derive this easily using the formal EM formulation; however, just using the soft counts is fine, as well.

(b) Generally, a choice that has lots of zeros will be bad. Other choices that depend on a lot of symmetry in the data will also cause problems. This model is not convex by a long shot, so there are plenty of local extrema; it is nontrivial to find a nontrivial initial setting, but a trivial one is fine here.

(c) My update to the code was this:

LM:

```
LM(m2m1,english(i,j)) = LM(m2m1,english(i,j)) + 1;
 .  .  .
LM(m2m1,i) = LM(m2m1,i)/LMc(m2,m1);
```

EM updates:

```
for j=1:m
  a = [];
  for i=1:l
    a(i) = T(deutsch(idx,j),english(idx,i)) * D(indexpack(j,l,m),i);
  end
  a = a / sum(a);
  for i=1:l
    Tn(deutsch(idx,j),english(idx,i)) = Tn(deutsch(idx,j),english(idx,i)) + a(i);
    Dn(indexpack(j,l,m),i) = Dn(indexpack(j,l,m),i) + a(i);
  end
end
```

normalization:

```
T = sparse(de_vocab, en_vocab);
D = sparse((mmax+1) * 50 * 50, lmax);
for enword=1:en_vocab
  nfactor = sum(Tn(:,enword));
  for deword=find(Tn(:,enword))
    T(deword, enword) = Tn(deword, enword)/nfactor;
  end
end
Dnfactor = sum(Dn,2) * sparse(ones(1,size(D,2)));
idxpack = find(Dnfactor);
D(idxpack) = Dn(idxpack)./Dnfactor(idxpack);
```

Here is the output I got:

```
Dear Klaus !
I have you . locked a small room .
You can read the books .
It
```

Some problems with this is that it tends to be somewhat sensitive to the start point; random was not the best choice, but it is ok. A better choice is to choose uniform for D and to use co-occurrence counts for T. In practice, one would train a simpler model (IBM Model 1) and transfer the probabilities. Also, a NULL word on the English side would help to "explain" common particle words in the German side. The last sentence is seven words long in German; this size does not appear in the corpus, so the translation is ridiculous. All in all, this method has many limitations, but considering its simplicity, the results are quite good and using this type of method is very appealing.

(d) (i) It will reorder more freely.

(ii) It will reorder less.

(iii) This is (essentially) the same as (i). If there are spots that tend to have attractive words on the English side (words that explain a lot of the German words), then all of the words will try to reorder there. In general, however, it will have a smoothing effect on the alignments.

2. (a) **4 points** If the kernel is defined so as to encode a high covariance between two points x_1 and x_2, then the observations at those two values should be highly correlated (i.e. should have roughly similar values). Thus, if the covariance between neighboring points is high along a long stretch of the x-axis, the function value will remain roughly constant along that stretch.

Using this intuition, we have the following set of matches: 1-d, 2-b, 3-a, 4-c. The kernels can also capture periodicity. For example, kernel (3) indicates that points which are $t - t' = 2$ apart will have low covariances while points which are $t - t' = 4$ will have higher covariances (i.e. roughly similar values). This introduces periodicity in the function values, as observed in Fig (a). Also, the $(tt')^2$ component of kernel (4) implies that the covariance between neighboring points increases with t, resulting in the curve moving in a single direction.

(b) **5 points** The log-likelihood of observing a set of values $y_1, \ldots, y_r$ at time-points $t_1, \ldots, t_r$ is simply the probability of sampling the r-dimensional vector $(y_1, \ldots, y_r)$ from $N(0, G)$ where G is the Gram matrix corresponding to the time-points. The code looks as follows:

```
function ll = log_likelihood_gp(params, t, Yobs)

%Gram matrix
```

```
G = ...

Ginv = inv(G);
detG = det(G);

ll = zeros(q,1);
for i=1:q
  y = reshape(Yobs(i,:), r, 1);
  a = 0.5*(y'*Ginv*y);
  b = (r/2) * log(2*pi) + (0.5*log(detG));
  l = -a - b;
  ll(i) = l;
end
```

(c) **7 points** The Expectation code is shown below. Recall that W_{ij} is the probability of gene i belonging to cluster j.

```
function W_new = Expectation(t, Y_obs, k, W, V)
P = sum(W);
P = P/sum(P);

[n,T] = size(Y_obs);
W_new = zeros(n,k);

for j=1:k,
  % get likelihood
  ll = log_likelihood_gp(V(j,:), t, Y_obs);
  % get posterior assignment probability
  W_new(:,j) = ll + log(P(j));
end

%normalize posterior
  % a more sophisticated handling of very small
  % number will be better, but the simplest
  % normalization method will suffice
  % for grading purposes

for i=1:n
    W_new(i,:) = exp(W_new(i,:)) / sum(exp(W_new(i,:)));
end
```

(d) **5 points** The best results are obtained with $k = 3$ clusters. With higher k, some of the clusters are either empty or two different clusters have similar curves. With lower k (e.g. $k = 2$), one of the clusters contains two different kinds of curves.

(e) **4 points** There are a few different ways of dealing with missing values. One approach would be to use the function curves estimated in the previous iteration of EM to compute the expected value of gene's expression at the missing time-point (in the first iteration, we could just use the mean of the adjacent, non-missing values).

Here's an alternative approach. Recall that the observations $y_1, \ldots, y_r$ play a role in our computation only when we compute the likelihood of sampling that set of values from a multi-variate Gaussians. Clearly, if some values are missing, this probability can be computed on a reduced set. In such a case, we simply use whatever values are available (i.e., the non-missing values) to construct the Gram matrix and then evaluate the likelihood of seeing that set of values. The Gram matrix in such a case would have a smaller size.

For example, if there are 10 time-points in total, you'd typically construct a 10x10 Gram matrix and then evaluate the probability of sampling a particular 10-dimensional vector from the corresponding 10-dimensional Gaussian. If, on the other hand, you only have 8 points, you'd construct a 8x8 Gram matrix and then evaluate the probability of sampling a particular 8-dimensional vector from the corresponding 8-dimensional Gaussian.

(f) **Optional: 4 points** There are many possible ways to guess an initial estimate of the gene clusters (for grading purposes, any reasonable approach is fine). Here we describe one such approach:

- Between each pair of genes g_1 and g_2, define a distance measure. This distance measure may be the Euclidean distance between the observations: $\sqrt{\sum_{i=1}^{r}(y_{1i} - y_{2i})^2}$. Another measure might be the Pearson correlation between these sets of observations.
- Using these pairwise distances, perform Hierarchical Agglomerative Clustering between the genes, i.e., start with each gene as a singleton cluster and at each step, merge the two most similar clusters. For example, if we perform *complete linkage* clustering, the distance between two clusters will defined as the largest distance between any pair of genes, one from each cluster.

(g) **Optional: 6 points** For notational convenience, we define the marginal likelihood for the case of a single cluster. The generalization to multiple clusters is straightforward.

Suppose there are r timepoints: $t_1, \ldots, t_r$ and let $f(t)$ be the cluster mean curve. Then, for any gene, the probability of seeing the observations $[y(1), \ldots, y(r)]$ in the cluster is the probability that each $y(l)$ is sampled from the Normal distribution $N(f(t_l), \sigma_n^2)$:

$$\prod_{l=1}^{r} N(y(l); f(t_l), \sigma_n^2), \quad \text{or}$$

$$\prod_{l=1}^{r} \frac{1}{2\pi\sigma_n} \exp(-\frac{(y(l) - f(t_l))^2}{\sigma_n^2})$$

The prior probability of seeing a cluster mean curve $f(t)$ is given by the probability of sampling the r-dimensional vector $\mathbf{f} = [f(t_1), ...f(t_r)]$ from the r-dimensional Gaussian formed by using the Gram matrix constructed from $t_1, \ldots, t_r$.

$$N(\mathbf{f}; \mathbf{0}, G) \tag{1}$$

where G is the Gram matrix as per the fixed GP.

The marginal likelihood is then:

$$\int \left[\prod_{l=1}^{r} N(y(l); f(t_l), \sigma_n^2) \right] N(\mathbf{f}; \mathbf{0}, G)\, d\mathbf{f}$$