COS 424
Homework #3
Due Tuesday, March 30th

See the course website for important information about collaboration and late policies, as well as where and when to turn in assignments.

Program and data files

This homework uses again the Reuters21578 dataset. Please download the file "`hw3data.tar.gz`" which is available from `http://www.cs.princeton.edu/courses/archive/spring10/cos424/w/hw3`. This archive contains the following files:

- `reuters_train.txt`, `reuters_test.txt` contain the preprocessed training and testing sets. Although we still use the "ModApte" split, these files are not exactly the same as those you were supposed to prepare during homework 2. Each line represents a single document and contains the numerical document ID followed by a sparse vector containing the number of occurences of each term in each document. These counts are not normalized.
- `reuters_dict.txt` provides the association between term index and the term themselves. Each line contains the term index, the term string, the total count of occurrences of this term in the training set, and the number of training documents containing the term. We have discarded all terms that were less than four characters long or appeared in less than three training documents. The total number of terms should therefore be slightly lower than the number of terms in the homework 2 files.
- A couple C++ files and a `Makefile` contain the almost complete implementation of two programs named `umix` and `humix`. The main source files are `umix.cpp` and `humix.cpp`. Part of the homework consists of completing and running these programs.
- The ancillary files `vector.h`, `vector.cpp`, `util.h`, `util.cpp`, and `wrapper.h` contain utility routines that you do not need to modify. The appendix of this document provides a brief description of the vector classes implemented by files `vector.h` and `vector.cpp`.
- File `homework.tex` contains the LaTeX source of this document.

Part 1 – Unigram mixture model

We wish to group the Reuters documents into semantically meaningful clusters that we call *topics*. We plan to achieve this using a mixture model whose k components represent k distinct topics. Let random variables X and Y represent a document and its topic. The unigram mixture model can then be written as

$$\begin{aligned} P_\theta(X) &= \textstyle\sum_{y=1}^{k} P_\theta(Y=y) P_\theta(X|Y=y)\,, \\ P_\theta(Y=y) &= \lambda_y \\ P_\theta(X=t_1t_2\dots t_L|Y=y) &= \beta_L\, p_{yt_1}\, p_{yt_2}\, \dots\, p_{yt_L}\,. \end{aligned}$$

In the above expressions, $t_1t_2\dots t_L$ represents the sequence of terms in the document X. The parameters θ of the model are (i) the prior probabilities λ_y of each component, (ii) the probabilities β_L that a document contains L terms, and (iii) the probability p_{yt} of occurrence of term t for component y. These parameters satisfy the normalization constraints $\sum_{y=1}^{k} \lambda_y = 1$, $\sum_{L=1}^{\infty} \beta_L = 1$, and $\forall y\ \sum_t p_{yt} = 1$. This model makes the assumption that the document length does not depend on the document topic because we use the same parameters β_L for all components. We do this because our model would too easily cluster documents by length instead of by content.

Question 1a Show that the log likelihood $\log L(\theta) = \sum_i \log P_\theta(X = x_i)$ can be written as the sum of an expression that depends only on the β_L parameters and an expression that depends only on the λ_y and p_{yt} parameters.

These two parts of the log likelihood can be optimized separately. Since we are not interested in modeling the document length, we can simply ignore the β_L and simply write

$$P_\theta(X = x_i | Y = y) \;=\; \prod_t p_{yt}^{\,n_{it}}\,,$$

where n_{it} represents the number of occurrences of term t in document x_i.

The provided C++ program `umix` maximizes the likelihood of this model using the Expectation Maximization algorithm. The main source code `umix.cpp` contains numerous comments. However there is a critical expression missing in the middle of the M step of the EM algorithm. The missing expression is clearly marked with text `INSERT_CORRECT_CODE_HERE`.

Question 1b Complete the missing expression in the M step. Provide a listing of that segment of the code showing your completion.

Here is the program output for computing a mixture with 5 components:

```
$ ./umix reuters_train.txt reuters_dict.txt 5 reuters_clus5.txt
Read 9603 documents
Read 7303 terms (maxid 7303)
Initializing 5 components
Pass 66 logl=-4.90179e+06
Saving components
0.339047  share offer issu corp debentur common dividend stock compani qtly
0.241021  loss profit billion year oper januari quarter februari rose from
0.186173  tonn wheat export market grain coffe crop usda sugar quota
0.137562  bank trade that brazil countri japan debt minist japanes foreign
0.0961973  mine barrel gold ship refineri crude said compani feet plant
```

The output summarizes each component on a line displaying its λ_y coefficient and its ten most distinctive terms.

Question 1c Run `umix` on the Reuters data for mixtures of 15 and 30 components. Can you recognize topics corresponding to some of the manual labels provided with the Reuters21578 dataset?

Question 2 – Hierarchical Clustering for UMIX

We now would like to organize the clusters hierarchically. Program `humix` follows an agglomerative clustering strategy. It starts from the topics computed by `umix` and repeatedly merges the two closest topics until all topics have been merged into a single component.

The distance between two topics is determined using a heuristic formula implemented by the function `mergeDistance()` in file `humix.cpp`. Given two topics g and j, the function first computes the term distribution p_{*t} of the merged topic

$$p_{*t} = \frac{\lambda_g p_{gt} + \lambda_j p_{jt}}{\lambda_g + \lambda_j}$$

and returns

$$\lambda_g H(p_{gt}, p_{*t}) + \lambda_j H(p_{jt}, p_{*t})$$

where the Hellinger distance is

$$H(p_t, q_t) = \sum_t \left(\sqrt{p_t} - \sqrt{q_t}\right)^2 .$$

Program `humix` also computes the likelihood on both the training set and a validation set. This is achieved by function `computeLogLikelihood()` in file `humix.cpp`. However this function lacks a couple critical expressions clearly marked with text `INSERT_CORRECT_CODE_HERE`.

Question 2a Complete the missing parts of function `computeLogLikelihood()`. Provide a listing of that segment of the code showing your completion. There are strong similarities between this code and the E step of the `umix` program.

Here is the beginning of the program output for the 5 components mixture:

```
$ ./humix reuters_train.txt reuters_test.txt reuters_dict.txt reuters_clus5.txt
Read 9603 documents
Read 3299 documents
Read 7303 terms (maxid 7303)
Read 5 components
----------------------  components: 5
Train logL = -4.90179e+06
Valid logL = -1.5112e+06
  #0: 0.339047  share offer issu corp debentur common dividend stock compani qtly
  #1: 0.241021  loss profit billion year oper januari quarter februari rose from
  #2: 0.186173  tonn wheat export market grain coffe crop usda sugar quota
  #3: 0.137562  bank trade that brazil countri japan debt minist japanes foreign
  #4: 0.0961973  mine barrel gold ship refineri crude said compani feet plant
Merging #2 and #3 as #1
----------------------  components: 4
Train logL = -4.93902e+06
Valid logL = -1.51722e+06
  #0: 0.339047  share offer issu corp common debentur stock qtly bond april
  #1: 0.323735  tonn export bank trade that market countri wheat brazil minist
  #2: 0.241021  loss profit year billion oper januari quarter februari rose from
  #3: 0.0961973  mine barrel gold ship refineri said crude feet plant grade
Merging #1 and #3 as #0
----------------------  components: 3
Train logL = -4.97703e+06
Valid logL = -1.52663e+06
...
```

Question 2b Run `humix` on the 15 and 30 components mixtures from question 1c. For each of these runs, produce a plot of the training and validation log likelihoods as a function of the number of components. Clearly mark the "elbow" if you can see it.

Question 2c Could we make `humix` run faster? How?

If you are interested by topic models and want to learn much more sophisticated approaches, you should definitely discuss with Sean Gerrish and see Professor David Blei. . .

Appendix – Vector classes

The provided source code relies extensively on two vector classes defined by files `vector.h` and `vector.cpp`. Class `FVector` and `SVector` respectively represent a dense vector and a sparse vector. Both classes define operators `<<` and `>>` to read and write a textual representation of the vector. This is useful for loading and saving the data files. Both classes provide a collection of nearly identical functions:

`x.size()`	Return the size of vector `x`.
`x.get(i)`	Return the `i`-th coefficient of vector `x`.
`x.set(i,v)`	Set the `i`-th coefficient of vector `x`.
`x.add(s)`	Add scalar `s` to vector `x`.
`x.add(y)`	Add vector `y` to vector `x`.
`x.scale(s)`	Multiplies all coefficients of `x` by `s`.
`dot(x,y)`	Return the dot product of vectors `x` and `y`.
`combine(x,a,y,b)`	Return the vector $a\mathbf{x} + b\mathbf{y}$.

When the index is larger than the vector size, function `x.get()` returns 0 and function `x.set()` automatically increases the vector size. You can also use the standard bracket syntax `x[i]` to quickly access the `i`-th coefficient of a `FVector x`. You must however be certain that the vector size is larger than `i`.

Sparse vector are implemented as a sequence of records corresponding to the nonzero coefficient. Each record contains the coefficient index and the coefficient value. Records are sorted by increasing index value. The following idiom can be used to iterate over all the nonzero coefficients of a `SVector x`:

```
for (const SVector::Pair *p = x; p->i >= 0; p++) {...}
```

In the body of the loop, `p->i` represents the coefficient index, and `p->v` represents the coefficient value.