CSCI1950-Z Cluster Assignment

March 27, 2009

1 Introduction

It's a tumor!

Cancerous cells generally exhibit very different gene expression patterns from normal cells. These differences can be used to distinguish tumor from normal cells, or to identify common subtypes of cancer. One method frequently used to identify such differences/group is to apply a clustering algorithm. In this assignment, you cluster gene expression data from hundreds of cancer samples.

You will implement the k-Means and CAST clustering algorithms. In addition, you will tweak the parameter arguments of these algorithms to determine the best result. Then, you will report your findings and justify a best value or rule of thumb for the parameter values.

2 Reading

Chapter 10, Jones and Pevzner. *http://site.ebrary.com/lib/brown/docDetail.action?docID=10225303*

3 k-Means Clustering

The k-Means algorithm you will apply is the one found on p. 348 J&P, "ProgressiveGreedyK-Means".

For the purposes of coding (see *Common Implementation*) I will divide the instructions into `init` and `step`:

- `init`: You will start by initializing the data structure(s) in which you will store the intermediate results and final results. You will derive the distance between elements in your input set by using a Euclidean distance function that we have defined in `CUtils.distance`.

  Another requirement is that you perform the partition P for use in the loop. For testing purposes, we require that you partition as follows: you initialize clusters of size n/k respecting the order of your input, except that the last cluster may have fewer than n/k elements when k does not divide the input evenly.

- `step`: Corresponds to the **while** forever loop.

  You must implement your own Δ function (for calculating the changes) and your own centroid function. For the centroid function, you should get the element's "vector" (an iterable list) by calling `myElement.getContents()`.

k-Means clustering does not make the use of any graph data structures!

4 CAST Clustering

The CAST algorithm is found on p. 354 J&P, "CAST(G,θ)".

- `init`: We are using a graph implementation called `JGraphT`. You will be passed an instance of `WeightedGraph` during the construction of your algorithm class. This will be the initially empty graph G, which you will populate with vertices by using the `distance` function. You can read *jgrapht.org* to acquaint yourself better with this package or use the links from the website.

  You will probably also want to initialize a data structure for S and P – just make sure that they are updated correctly as the algorithm runs.

- `step`: Corresponds to the **while** $S \neq \emptyset$ loop.

  Good implementations will use efficient methods of manipulating the clusters and elements. Remember to make sure that P has the correct structure for the result format (see *Common Implementation*).

5 Common Implementation

The way you will implement the algorithms is shared in common, as defined by `CAlgorithm`. Your two algorithms are the `CStudentKMeans` and `CStudentCAST` classes found in the `cs195z.cluster` package.

5.1 Required Methods

Each algorithm has a constructor that you are not allowed to change. However, you must implement the `init()`, `step()`, and `getResult()` methods. The `init()` method is run before execution of the loop, so it should perform your initial chores. The `step()` method represents one iteration of the while loop of your respective algorithm. Do not accidentally run all your iterations at once!

The `step()` method has a required return boolean value. **Note** that you must test for the termination condition at the end of each run of your `step()` method. Note the difference between the normal definition of **while** and this requirement. If the algorithm has completed its task, then you should `return false;`. Otherwise, you should `return true;` as the last line of the `step()` method.

After the algorithm terminates its operation, the GUI will call `getResult()` on your algorithm. Therefore, make sure that you assign a correct data type to `_result` before you `return false`.

5.2 Java Generics

Please don't be discouraged by the many Java generics in the support code. We make some requirements that your algorithm is generic for any `<T,E>` where `T` are the type of numbers used in the vectors and `E` is the tag for each element's (gene's) name. This makes your code reusable.

If you reference a generic element, you can copy `<T,E>` as necessary – where implied `<T extends Number,E extends Serializable>` – or even ignore generics (Java generics are a bit flexible). Don't worry if the compiler gives you warnings about unchecked casts – but do make sure your code works.

5.3 Others

Please make judicious use of `System.out.println`'s. This will help you a lot for testing parameter values (see *Parametrization*).

The step method (albeit awkward) lets you see the results of one computation (again, `println`) so that you can check them and/or the graph. Thus, you must store intermediate values to the private instance variables of your algorithm classes.

6 GUI Interface

To start, you click on one of the "Load" buttons. It will take in a parameter and a data file, then initialize your algorithm. To execute, either click "Step" for one iteration, or click "Complete" to let the algorithm go to completion. Although your algorithms are threaded, we only allow execution one at a time for collecting results – when your algorithm completes you can do it again. If you do not like selecting a new file each time, toggle "Use same file" and it will use the previous.

If you launch CAST, it will launch another frame that is the visualization of your distance graph. In either case, you should retrieve your results from the console.

7 List of Support Code

You write the following code, in the `cs195z.cluster` package. Comment appropriately and do not modify constructors.

- `CStudentKMeans`
- `CStudentCAST`

The support code in the `cs195z.support` package that is relevant to this project:

- `CAlgorithm` is the interface for your algorithm. Note that your class is constructed with a `List` of `CElements` but you return an `Iterable` of clusters, each of which is an `Iterable` of elements. `Iterable` is an interface implemented by any of the `Set` or `List` classes in `java.util`.
- `CAbstractAlgorithm` deals with threading code, and you need not concern yourself with it.
- `CDataParser` parses the data files.
- `CElement` Holds the data corresponding to each data "point" or element. Note that it has a list (or "vector") of `Numbers` that you get by calling `getContents()`. Also it stores a name (any serializable class) with `get/set` methods. You may construct a CElement using one of the two methods.
- `CUtils` contains the GUI interface and data parser. Also, it has the methods `distance` and `printClusters` that might be useful, or good code examples. We use `printClusters` at termination of your algorithm.

Data is found in the `FILES_ROOT/data` folder (see *Data Sets*).

8 Data Sets

You will test your program on four datasets:

- `goodrandom.txt` – A small file where you should find clusters (positive control).
- `badrandom.txt` – A small file of randomly generated N(0,1) values. You may find clusters, but they should be unsurprising.
- `alldata.txt` – This is the gene expression data from 381 patients with brain cancer. This data was produced as part of a a large project called The Cancer Genome Atlas (*http://cancergenome.nih.gov/*). The entries in this dataset are (unnormalized) measurements of gene expression from a microarray.
- `subset.txt` – A subset of the file above. Your analysis will refer to this file, but try the `alldata.txt` challenge too!

You may make your own data files, but do respect the file format. The first line should have "Name" then tab, then a number, then tab, etc. for every column of your matrix. The parser will match up the data to your first line, so make sure that each row respects the columns defined above.

9 Parametrization

The third part of the assignment (after the algorithms) is to determine the best k or θ parameters, respectively. This is a so-called "black art."

You will perform the following tasks as part of the README, which we prefer in a pdf format.
ALL OF THE FOLLOWING ARE TO BE PERFORMED ON THE `subset.txt` DATA SET!

- For k-means, run the algorithm on all k, $1 \leq k \leq n$ where n is suitably large. Plot k versus the squared error distortion (summed for each cluster center) on p.346. (Your plot should justify both your optimal k and your choice of n.)
- Let d_{min} and d_{max} be the minimal and maximal pairwise distances in the entire data set. For CAST, run the algorithm over fifty equally spaced values of θ that span the range $[d_{min}, \frac{d_{min}+d_{max}}{5})$. Again, plot θ versus the squared error distortion.
- Determine the optimum value of k and θ for the data set. Write a paragraph (≤ 7 sentences) justifying your choices. At least refer to your plots, but also any other metrics as necessary. The best responses will be concise but convincing.
- Compute $H_{ave}, H_{max}, S_{ave}$, and S_{min} for each of the algorithms (at the optimal parameter value for that algorithm). (You need not program this). Ben covered these metrics on the whiteboard Wednesday before break; if you need help contact a TA.
- Write 3-4 sentences comparing the best outputs of the k-means and CAST algorithms and explaining the significance of the above metrics. Which is "better"?

10 Code Acquisition and Running

The code may again be acquired from the public directory `/course/cs195z/pub`. We are maintaining the same structure so that the course projects are unified. One way of acquiring the code is to do

```
cp -r /course/cs195z/pub MY_CS195Z_DIR/
```

You shouldn't overwrite your implementation of `PStudentTreeAlgorithms`. All other files may be overwritten, although note that `cs195zlib.jar` has not been modified.

Once you are in your local cs195z directory, just run

```
ant cluster
```
OR `ant -Dout=myfile.txt cluster`

The second option will save the console output to a file. The previous ant targets are still available.

11 Evaluation

Evaluation is performed on the following four bases:

1. Correctness (50%) – The algorithm works on arbitrary (but valid) test data sets and provides in each case a minimum parsimony score or maximum likelihood assignment solution (objective).

2. Logical adherence to algorithm (30%) – We should be able to clearly identify how your code performs precisely the expected algorithm. We take into consideration appropriate commenting, good structure, and no introduced inefficiencies.

3. README & Analysis (20%) – Must exist and follow the specification of part 9. Also include any bugs/concerns.

4. Extra-credit (up to 6%) – There are many opportunities for extra-credit. Anything additional to the specification counts: implementing correlation distance, permuting the rows of the test matrices and analyzing the variance of the algorithms, extending the GUI, or adding additional functionality or parameters beyond the J&P algorithms. If you do one thing to our satisfaction, you get 3%; for two 5%; for three or more 6%.

12 Good Luck!

The assignment is due Wednesday, April 15 at 11:59pm. Please email the minimal source code you wrote and README/plots (pdf please) to `cs195ztas@cs.brown.edu`.