CS 75/175 | QBS 175 (Bioinformatics), Winter 2017

Homework 2, due Tuesday, Jan 24

This homework explores basic unsupervised and supervised analysis of expression data. You are to implement your own code for the various function bodies, not just call code from machine learning packages.

Submit on Canvas, before class, your solutions to the following. For the programming problems, submit a Python file with your code inserted in the provided template. For the application part submit a PDF, Word file, etc.

Programming

Expression data will be represented as a list of `ExpressionProfile`s, each for a different sample, with an array of values for different genes.

1. Implement hierarchical clustering for a given set of expression data. An `ExpressionCluster` class represents binary trees: leaves have `ExpressionProfile`s and inner nodes have left and right children (themselves `ExpressionCluster`s). Use Euclidean distance to measure profile-profile distances. To measure cluster-profile and cluster-cluster distances, allow average linkage (i.e., the average of all pairwise distances from members of one cluster to members of the other) or min or max linkage (i.e., the smallest or largest distance from a member of one cluster to a member of the other).

   The number of profiles is fairly small, so you need not worry about efficiency in constructing the tree, and can just repeatedly find the closest pair among all remaining ones. You might want to store the pairwise distances, though, in order to easily compute distances for a new cluster from distances involving its children.

2. Implement a k-nearest neighbors classification algorithm. k-nn is one of the simplest classifiers, but can be efficient and good. Its name basically gives the algorithm: given a test case, find its k nearest neighbors in the training set, and have them vote on the label for the test case. The most popular label among the neighbors is selected. Use Euclidean distance between profiles to select the neighbors.

3. Cross-validation is an important aspect of assessing generalization ability of a classifier. Implement repeated n-fold cross-validation, in which for each repetition (say 10 of them), the data is split into a given number (say 5) of "folds" (subsets). For each fold, use it as the test set (make predictions for its labels) based on a training set comprised of the members of the other folds. In this case, that means for each expression profile, predict its label according to votes from its nearest neighbors in the other folds. For each reptition, randomly partition into new folds, thereby varying the training data used for predicting each instance. Assess the quality of the classification in terms of how many of the test instances are assigned correct labels, keeping separate counts for each label (as the numbers can be quite imbalanced).

4. Finally, permutation testing allows assessing whether we could do as well with any (bogus) breakdown of profiles into classes, or if the data really does support much better classification of these particular labels. With "wide" data (many more features than samples), it's increasingly likely that we could stumble upon features that just happen to be discriminative, regardless of whether there's any real

biology there. To support permutation testing, implement a function to shuffle up the labels of the profiles, returning a new set of profiles that can be subjected to the above methodology to provide a baseline for performance comparison.

Application

Provided are a simple concocted example for development, along with the Golub et al. ALL/AML dataset discussed in class (*Science* 1999), and a cancer recurrence dataset (from van't Veer et al., "Gene expression profiling predicts clinical outcome of breast cancer", *Nature* 2002). Feel free to find others; please describe and upload.

Provide a *short* document presenting and discussing the output of your functions on these two datasets and senstivity to the parameters / choices. (Note that `plt.savefig` can be used to output the current figure to a file.)

In particular:

1. How "pure" are the hierarchical clusters on the original data, both in terms of visual patterns and the labels in the subtrees? How does the choice of linkage affect that?
2. How well do you classify with different choices of k? How does the number of folds affect that?
3. How well do you classify under relabeling? (Do the permutation multiple times since some will be better and some worse.)

Extra credit

See the course syllabus regarding the use of extra credit; please consider it as fun and educational stuff to do (possibly) after you've gotten the main assignment in tip-top shape. Here are some ideas; your own novel extra work will also be considered.

- Implement another clustering algorithm, e.g., k-means, SOM, etc.
- Select informative features within the training set, e.g., those that are most different between the classes.
- Implement another simple classifier algorithm (again, not just with single-line calls to a package ;).

Provided materials

hw2.py Template and some basic functionality, including the classes discussed above. Note that there is a simple CSV-based reader, which assumes that a ':' separates the id from the label (if there is one), and a ',' separates each of the values. A plotting function generates a pseudocolor plot of the profiles.

As before, I strongly encourage you to develop interactively in `ipython`, playing around with the functions in the interpreter as you're developing them.

```
ipython --matplotlib -i hw2.py
```

You may of course add additional functions and classes as useful. Please stick with the provided function signatures and command-line driver. If you need to provide additional functionality, include default value arguments so that the TA can just as easily run your code.

tests.zip The datasets described above.

Grade breakdown

1. hierarchical: 30 total
   (a) pairwise profile distances: 8
   (b) find closest pair: 8
   (c) compute cluster distances: 9 (3 each for 3 linkages)
   (d) build up tree: 5
2. k-nn: 20 total
   (a) neighbors: 8
   (b) label according to votes: 7
   (c) whole test set: 5
3. cross-validation: 20 total
   (a) randomly split into folds: 7
   (b) test each fold, computing # correct predictions: 8
   (c) repeated, averaging: 5
4. relabel for permutation testing: 10
5. application: 20 total
   (a) clustering analysis: 8
   (b) k-nn cross-validation analysis: 8
   (c) permutation: 4