Institut für Informatik
Prof. Dr. Heiko Röglin
Magdalena Aretz

Lab Efficient Algorithms for
Selected Problems: Design, Analysis
and Implementation
Winter 2013/2014

Tasklist 4
Due Date: 16.12.2013

Basics

1. **Testdriven Development**:
   - Have a look at
     `http://en.wikipedia.org/wiki/Test-driven_development`.
     Think about what parts of your code it might be useful to implement tests for.
   - Recapitulate what you have learned from the first tasklist (from the *Eclipse and Java for Total Beginners* tutorial) about how to write JUnit testcases.
2. **GNU R**:
   - Learn about advanced concepts and data structures in `R` by working through chapters 4 – 6, 9 and 10.1 of
     `http://cran.r-project.org/doc/manuals/R-intro.html`.

Implement

1. Java: By now your code contains several useful methods that you might want to reuse in different occasions later on or even in the context of a different project (e.g. a method that computes the euclidean distance between two points, a method for writing a twodimensional array into a .csv file etc). Create a class *KMUtilities* and move those functions there.
2. Java: Write JUnit testcases for all methods in *KMUtilities*. For example you could define two arrays and check whether the method for computing their euclidean distance works as expected.
3. Java: By now, you have some variants of the *initialization method* for the centers in `k-Means`. Now implement at least one modified *update rule*. Here are some examples:
   - `Single-Point k-Means` – Only reassign one point per step.
   - `Hartigan's Method` – Only reassign one point per step and also take into account those points whose reassignment only improves the objective function during the next iteration of one E-Step plus one M-step.
   - `Lazy k-Means` – Only reassign points if the distance to the new center is at least by some ϵ smaller than the original value.

- `Jump k-Means` – Introduce a *jump*-step in which centers that moved less than some ϵ-value during the M-step are pushed into a random direction. In order to guarantee termination, the *jump* step should be reversed if it did not improve the objective function.
- ...

   You are free to pick any other variant you already know about. If you want to, you can even think of a new modified update rule.

4. Java: Use your existing code to conduct a broader series of experiments. Make sure you are able to combine every choice of data, preprocessing operator, choice for k, initialization rule and update rule into one setting. For each setting you should obtain at least two different performance metrics: the final error value and the number of steps until convergence.

5. `R`: Try to capture the outcome of the experiments in different kinds of plots. Essentially you can plot every pair of parameter vs. performance metric, e.g.

   - different values for $k \to$ final error
   - choice of operator for preprocessing $\to$ number of steps until convergence
   - choice of the initialization method $\to$ final error
   - ...

   Try to obtain significant results and think about explanations for what you measured.

6. Think about one central question you want to answer in your final presentation. For example:

   - How does the structure of the data influence the performance of `k-Means`?
   - How does the variance of the final error of `k-Means` change with the method of initialization?
   - To what extend does preprocessing (scaling, standardization, outlier detection etc.) influence the running time and final error of `k-Means`?
   - ...

   You should be able to explain why this question is interesting and what kind of experiments could answer it. You should also already think about how to visualize the behavior you might observe.