jClustering developer manual

José María Mateos
jmmateos@mce.hggm.es

August 6, 2013

Contents

1 Introduction

This document explains how to develop new clustering algorithms using the jClustering API. If you just want to use this software, please refer to the user manual (`https://github.com/HGGM-LIM/jclustering/blob/master/doc/user_manual.pdf?raw=true`).

This guide expects the developer to be familiar with the ImageJ class structure, as the basic concepts will not be explained here. For more details regarding plugin development under ImageJ, please refer to `http://imagingbook.files.wordpress.com/2013/06/tutorial171.pdf`.

Starting from version 1.2.4, the latest API documentation is attached to each release. Please download that copy of the API as it is the most useful resource for developers, apart from this guide.

jClustering is offered as a Maven project from its main github page. It automatically downloads all the dependencies (except the fastICA library, which will need to be downloaded from its main page – check the user manual) and creates a `.jar` file.

2 General class architecture

The following image (figure 1) is a reproduction of the one used in the PLOS ONE paper describing this tool.

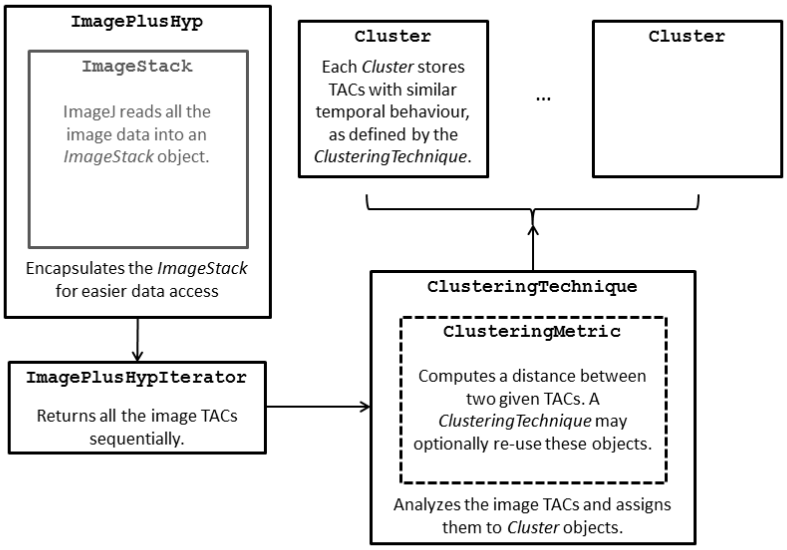

Figure 1: jClustering class structure and relationships.

We can sum up briefly what the presented class structures wants to achieve. As this tool is intended to be used in dynamic images (2D + time or 3D + time), the `ImagePlus` object will contain a `ImageStack` will all the slices and

temporal information (ImageJ uses a 3D matrix to store 4D images). As there are no immediate methods to retrieve the time-activity curve (TAC) for a given voxel (that is, the array of different gray scale intensities of that voxel through the time axis), the `ImagePlusHyp` object encapsulates the original `ImagePlus` and adds a `getTAC(int x, int y, int slice)` method that allows to access that data very easily.

Furthermore, to avoid the developer the worry of having to keep in mind the dimensions of the original image, this object implements the `Iterable` interface using the `ImagePlusHypIterator` class and returns, one by one, all the voxels in the image using objects of the class `Voxel`, which includes the original voxel coordinates and the TAC as publicly accessible class fields.

It is strongly recommended to use this iterator, as it will check whether a given voxel has been masked (the value is 0.0 through all the frames) and will not return it, diminishing the complexity of the clustering problem to solve[1]. In any case, you are free to use the `getTAC(int x, int y, int slice)` method at your own risk.

If you want to implement a new clustering technique using jClustering, it is very likely you only need to extend the `ClusteringTechnique` abstract class. This class provides the necessary methods for the clustering operation as well as references to the objects that will be used to represent the different clusters (via an array of `Cluster` objects). If you want instead to implement a function that computes the distance between two given TACs and that can be reused between different `ClusteringTechnique` objects, extend the `ClusteringMetric` class instead.

3 Implementing a `ClusteringTechnique`

The `ClusteringTechnique` abstract class contains internal references to the following objects:

- An `ImagePlusHyp` named `ip`. This is the main object you are going to read data from (typically via the provided `ImagePlusHypIterator`).
- A `ClusteringMetric` named `metric`. Depending on the type of algorithm that you are implementing, you might not need to worry about

[1] Utilities for masking dynamic studies have been published as part of the LIM tools ImageJ plugin available at `https://github.com/HGGM-LIM/limtools`. The particulars of this process are outside the scope of this document.

this reference. Currently, the only implementation that uses different distances is k-means.

- A `JPanel` named `jp`. If you need to implement some configuration options, this is the object that you need to use to add different GUI elements.
- An `ArrayList<Cluster>` named `clusters`. This object has already been initialized for you and you only need to add new `Cluster` objects to it.

There is only one method that you **must** implement in your algorithm: `process()`. This method must fill in the `clusters` object according to your local algorithm. Also, your class must belong to the `jclustering.techniques` package in order to be found by the automatic class detection algorithm (see section 5 for more information on this).

Check the code example shown on listing 1. It provides a very simple algorithm consisting on grouping together those voxels with TACs that peak at the same time.

Listing 1: Your first `ClusteringTechnique`.

```
package jclustering.techniques;

import jclustering.Voxel;
import static jclustering.MathUtils.getMaxIndex;

public class SampleTechnique extends ClusteringTechnique {

    @Override
    public void process() {
        for (Voxel v : ip) {
            // Find the maximum index
            // +1, min_cluster = 1.
            int n = getMaxIndex(v.tac) + 1;
            // Add this voxel to its cluster
            addTACtoCluster(v, n);
        }
    }
}
```

This code uses the `getMaxIndex(double [] tac)` method from the `jclustering.MathUtils` class, which provides misc mathematical utilities. This method returns the index of the `tac` array that contains the maximum value. With this value

(plus 1, as there is no cluster 0), we can call the method `addTACtoCluster(Voxel v, int n)`, that automatically takes care of the rest of the cluster addition process.

In case we want to add a selector in order to use some of the existing metrics (which we can then access through the `metric` object), we only need a little extra bit of code (check Listing 2) as there is an `addMetricsToJPanel(JPanel jp)` method that takes care of most of the operation. Other GUI components must be fully implemented to work (though there are several static helper methods in the `GUIUtils` class.

Listing 2: A `ClusteringTechnique` that includes a `ClusteringMetric` selector.

```
package jclustering.techniques;

import java.awt.event.ItemEvent;
import javax.swing.JPanel;
import jclustering.Voxel;
import static jclustering.MathUtils.getMaxIndex;

public class SampleTechnique extends ClusteringTechnique {

    @Override
    public void process() {
        for (Voxel v : ip) {
            // Find the maximum index
            // +1, min_cluster = 1.
            int n = getMaxIndex(v.tac) + 1;
            // Add this voxel to its cluster
            addTACtoCluster(v, n);
        }
    }

    @Override
    protected JPanel makeConfig() {
        JPanel jp = new JPanel();
        addMetricsToJPanel(jp);
        return jp;
    }

    @Override
    public void itemStateChanged(ItemEvent arg0) {
        super.itemStateChanged(arg0);
    }
```

```
}
```

3.1 Returning additional information

After the `clusters` object has been correctly filled, jClustering takes care of everything else and saves the results to a file and shows the resulting clusters on screen (please refer to the user manual for more information on this). However, this approach might not be enough for all the information that we want to provide after our algorithm finishes.

Consider the PCA `ClusteringTechnique` implementation (`https://github.com/HGGM-LIM/jclustering/blob/master/src/main/java/jclustering/techniques/PCA.java`). This technique shows an additional image on screen; it uses the very specific `RealMatrix2IJ(RealMatrix rm, int [] dim, ImagePlusHyp ip, boolean skip_noisy, String name)` method, but any `ImagePlus` object can be created and shown on screen at any point during the `process()` implementation.

However, it also stores in a file additional information (in this case, the values of the principal components found). Internally, the `ClusteringTechnique` class also contains the object `String [] additionalInfo`, which defaults to a `null` reference. If this algorithm needs to save on file some extra numerical values, this is the string that needs to be used. Consider the following code sample from the `PCA.java` file (code listing):

Listing 3: Code sample that allows to save extra information

```
// Fill in the additionalInfo array.
additionalInfo = new String[2];
additionalInfo[0] = "pca_vectors";
StringBuilder sb = new StringBuilder();
int rows = svdv.getRowDimension();
for (int i = 0; i < rows; i++) {
    double [] row = svdv.getRow(i);
    sb.append(Arrays.toString(row));
    sb.append("\n");
}
// Remove brackets
String temp = sb.toString();
temp = temp.replace("[", "");
temp = temp.replace("]", "");
additionalInfo[1] = temp;
```

This string array is expected to be of even length. The i positions contain the initial name of the file (it will be completed using a timestamp and the `.txt` extension) and the $i + 1$ contain the actual information that will be saved to that file.

4 Implementing a `ClusteringMetric`

A `ClusteringMetric` is used inside a `ClusteringTechnique` that calls internally the methods of its `metric` object. It is a very simple object used to compute the distance between two given TACs while at the same time allow for future code reusing. Currently, only the k-means implementation allows to use different metrics.

The only method that **must** be implemented is `double distance(double [] a, double [] b)`, which returns the distance between TACs `a` and `b`. There is a `void init()` method that can be used as a constructor to initialize the objects that will be used in each `distance` call.

As a sample, let's take a look at the correlation implementation (`https://github.com/HGGM-LIM/jclustering/blob/master/src/main/java/jclustering/metrics/Correlation.java`) in code listings 4.

Listing 4: `Correlation` metric.

```
package jclustering.metrics;
import java.util.Arrays;
import org.apache.commons.math3.stat.
        correlation.PearsonsCorrelation;

public class Correlation extends ClusteringMetric {

    private PearsonsCorrelation pc;

    @Override
    public double distance(double[] centroid, double[] data)
    {

        double corr;

        if (Arrays.equals(centroid, data)) {
            // Same contents, do not compute the correlation
            corr = 0.0;
        } else {
```

```
            // Turn a correlation score into a distance
            corr = 1.0 - pc.correlation(centroid, data);
        }

        if (!Double.isNaN(corr)) return corr;
        else                     return Double.MAX_VALUE;

    }

    @Override
    public void init() {
        pc = new PearsonsCorrelation();
    }

}
```

As you can see, this implementation uses the `init()` method to initialize the `PearsonsCorrelation` object that is used in the calls to `distance`. This method is called just once when the `ClusteringMetric` is initialized inside the `ClusteringTechnique`.

5 Automatic class detection

Techniques and metrics are added automatically to their respective selectors thanks to class autodetection methods (`getClusteringTechnique` and `getClusteringMetric` in `Utils.java` (`https://github.com/HGGM-LIM/jclustering/blob/master/src/main/java/jclustering/Utils.java`)). This allows new developers to implement directly their own classes and they will be available from the main window the next time jClustering is run.

6 Behavior of the `Cluster` object

The `Cluster` object behaves in two different ways depending on how it is initialized. This reflects the two different ways it may work during an iterative clustering process:

1. The centroid for the cluster may be fixed at creation time and does not change when a new voxel is added.

2. The centroid for the cluster is modified with each voxel addition.

The first behavior is characteristic of k-means, for instance, while the second one belongs to other approaches such as leader-follower. In order to avoid implementing two different `Cluster` objects, it was decided that the constructor would define how this object behaves.

1. If the constructor is called with a `double []` parameter that represents the centroid TAC, it will be fixed.
2. If the constructor is called with no parameters or with a `Voxel` object, the centroid will be modified with each addition, computing the new mean TAC.