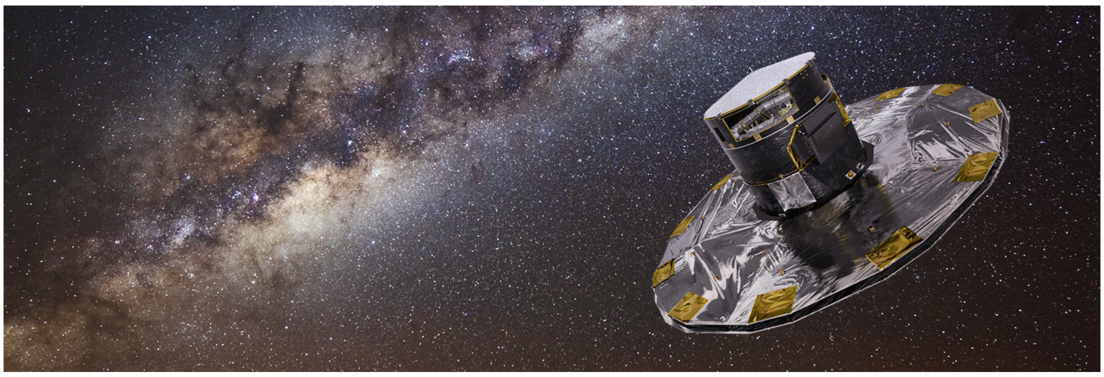


Machine learning methods to classify objects observed by the Gaia space telescope

Dr. Nick Rowell

Introduction

The Gaia space telescope is an ambitious mission by the European Space Agency to determine the structure, formation and evolution of the Milky Way galaxy. This is achieved by measuring the precise positions, motions, chemical compositions and other properties for around one billion individual stars in the Milky Way and nearby galaxies.

Gaia was launched in 2013 and has been in regular science observing mode for two years, observing roughly 50 million stars per day. For efficiency, on-board object detection locates each star and transmits back to Earth just a small thumbnail image. Along with the majority of single, isolated stars, these windows can contain cosmic rays, diffraction spikes, hot columns and many other things. A whole zoo of different kinds of object can trigger the onboard object detection, and it is important to be able to classify these into the various types in order to filter them appropriately in the data processing pipeline. There are far too many observations to classify manually, so an automated method is required. Machine learning algorithms, often used for face and speech recognition, are potentially very powerful tools to tackle this problem.

This document is an introductory guide to the software and data put together for this summer project.

Contents

1 Input data

Several types of data will be used in the course of this project. These are stored in files that can be loaded and processed using the Java programming language, and several examples of this are provided in the project source code. This section provides a description of the observations made by Gaia, and the types of data that are derived from these for use in this project.

1.1 Gaia observations

Gaia observes at optical wavelengths using regular CCDs (*Charge-Coupled Devices*), much like a digital camera. Gaia has 106 CCDs that are arranged in a large grid around one metre across (see figure 1), and the CCDs are divided among four different instruments that are designed to make four different kinds of observation. These are:

- The *Sky Mapper*, which is used to detect stars in real time onboard the satellite
- The *Astrometric Field*, which is used to make very precise measurments of the positions of stars
- The *Blue and Red Photometers*, which measure the brightness of stars at two wavelengths
- The *Radial Velocity Spetrograph*, which measures the spectrum of stars and is used to determine their velocity towards or away from Earth

This project will deal exclusively with data from the Astrometric Field instrument. Note that there are a couple of other instruments indicated in this diagram (the Basic Angle Monitor and Wavefront Sensor) that are not used to observe stars.

Gaia observes the sky in *drift-scan* mode, which means that it constantly spins like a top and takes images of stars as they move slowly across the CCDs from left to right over a period of around one minute. It takes 4.4 seconds for a star to cross one CCD. Stars first encounter the Sky Mapper, where they are detected automatically and their motion across the remaining instruments is predicted. This allows Gaia to transmit back to Earth only small patches of the image surrounding each star, rather than the entire image which would be too much data. Stars then drift onto the Astrometric Field, where they are observed by each of nine CCDs as they cross the whole instrument. Each observation produces a small image of the star that is 18-by-12 pixels in size. These nine observations are bundled up and transmitted back to Earth, and constitute the data that will be used in this project.

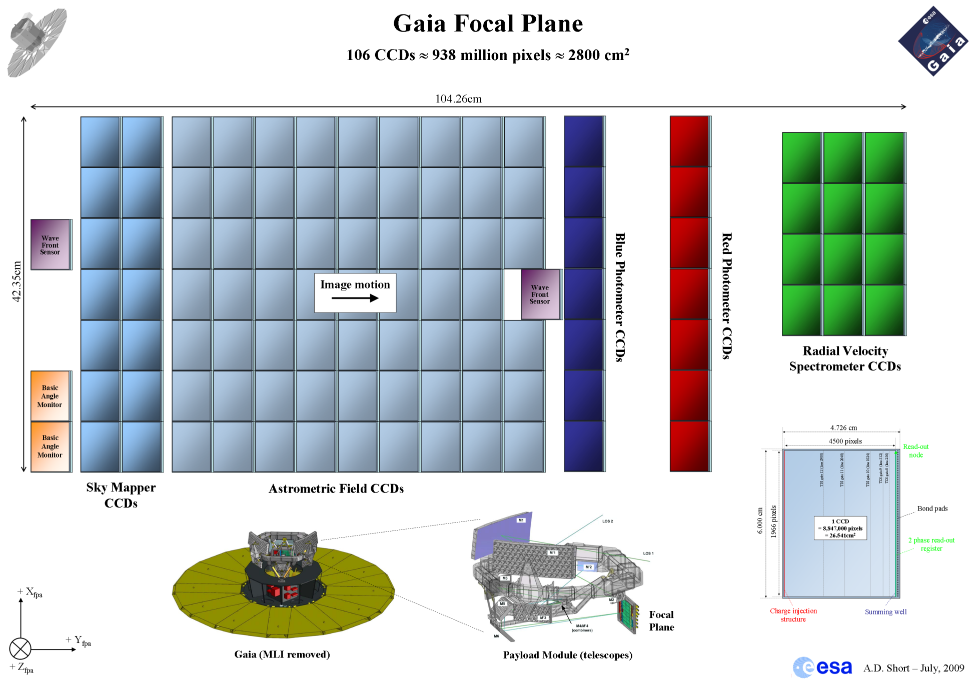


Figure 1: Diagram of the Gaia *focal plane array*; the arrangement of 106 CCDs that make up the different scientific instruments installed in Gaia.

1.2 Window.java

The raw Gaia observations have been pre-processed into a format that will be easier to handle within this project. Each 18-by-12 pixel image has been background-subtracted, de-biased and converted from digital units to number of electrons in each pixel. A Java class called `Window` has been created to represent the data for one of these images, as well as a few other bits of 'housekeeping' data that record which CCD the image came from, what time it was acquired, and a few other things that will become more relevant as the project progresses. The input data for this project therefore consists of files containing a large number of `Windows`, which can be loaded and processed within a Java program.

1.3 Source.java

`Windows` on their own are not very useful: we need to locate and extract objects within the windows in order to have something to classify. A single `Window` can contain a number of objects of different types (e.g. one or more stars and possibly cosmic rays), and it's important that we identify which pixels belong to which object in order to classify them seperately. This is the task of *Source Detection*, and it is carried out using existing code already developed for this project. A Java class called `Source` has been created that contains the data representing one detected object. This data includes a list of the pixels that belong to the source, a variable that records the type of object that the source has been classified as, and a set of statistics computed from the source pixels. These statistics quantify things such as the shape of the source (how round

Statistic	Description
Flux Ratio	Ratio of the largest pixel to the average of its neighbours
Eigenvalue 1	Largest eigenvalue of the flux-weighted pixel dispersion matrix
Eigenvalue 2	Smallest eigenvalue of the flux-weighted pixel dispersion matrix
Orientation	Orientation of the principle axes of the flux-weighted pixel dispersion matrix

Table 1: Description of the statistics used to classify each Source.

or elongated it is) and the flux of the source (whether it is broad and flat or sharply peaked). **The main goal of this project is to use machine learning methods to convert the source statistics into a classification.** For completeness, table 1 describes these statistics and how they are computed. However, bear in mind that these details are not important when using machine learning methods (so don't worry if you don't understand them all). One of the main benefits of machine learning is that the precise meaning of the inputs is not relevant.

In figure 2 four windows are shown along with the sources extracted from them (indicated with the red contours).

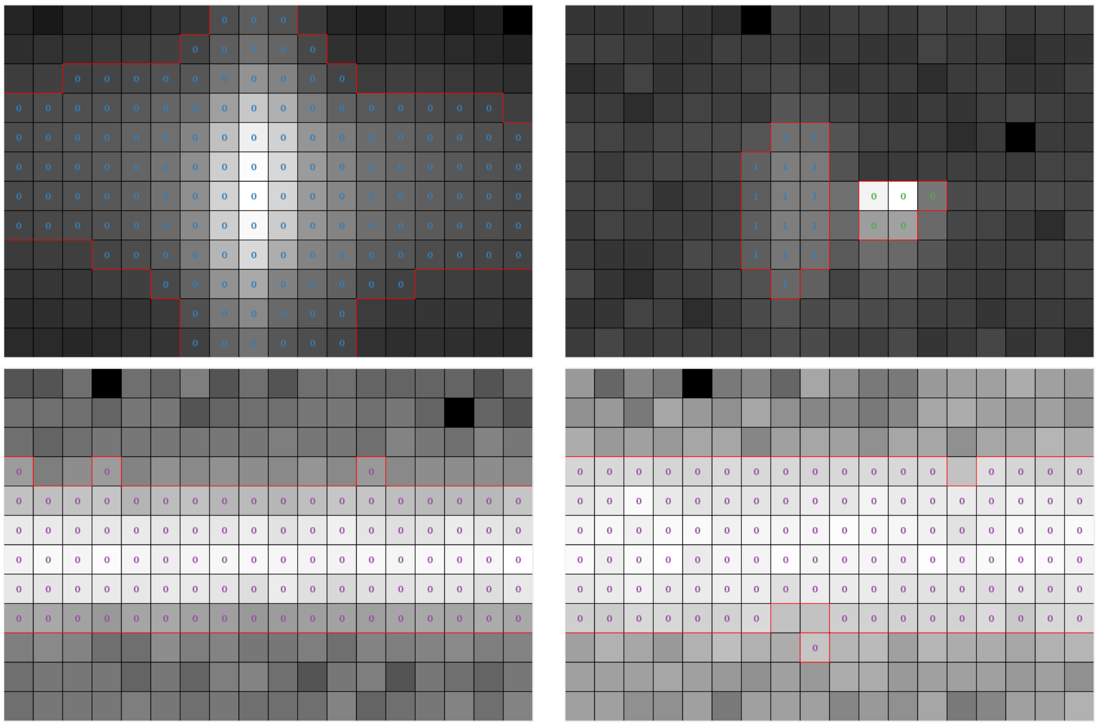
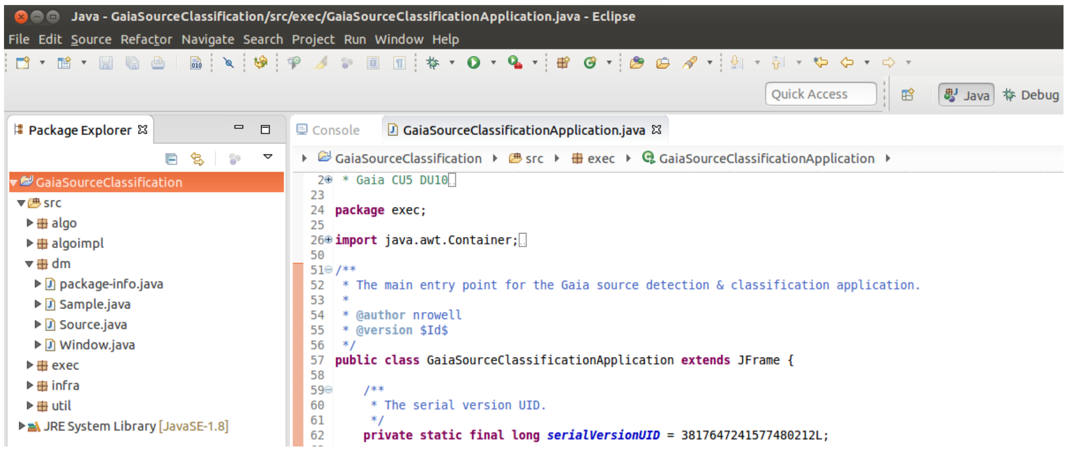


Figure 2: Visualisation of four windows, along with the sources extracted from them. In the top left window a single stellar source (a star) is present; in the top right window there's a star and a cosmic ray. The lower two windows show examples of diffraction spikes in the AL (ALong-scan) direction.

2 Software development environment

All of the software provided for this project is in the Java programming language. The software is packaged as a project in the Eclipse integrated development environment (IDE), so Eclipse must be installed and used to develop and run the software. The Eclipse IDE for Java Developers can be downloaded from `http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/marsr`.

An IDE such as Eclipse provides a supporting framework in which to write code, with features like syntax highlighting and project navigation that make it easy to develop software and keep track of everything. It encourages good coding habits and software design practise: generally if something is difficult to do in Eclipse then there's probably a better way to do it.


Java - GaiaSourceClassification/src/exec/GaiaSourceClassificationApplication.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Debug
Package Explorer
GaiaSourceClassification
src
algo
algoimpl
dm
package-info.java
Sample.java
Source.java
Window.java
exec
infra
util
JRE System Library [JavaSE-1.8]
Console
GaiaSourceClassificationApplication.java
GaiaSourceClassification
src
exec
GaiaSourceClassificationApplication

```
 * Gaia CU5 DU10

package exec;

import java.awt.Container;

/**
 * The main entry point for the Gaia source detection & classification application.
 *
 * @author nrowell
 * @version $Id$
 */
public class GaiaSourceClassificationApplication extends JFrame {

    /**
     * The serial version UID.
     */
    private static final long serialVersionUID = 3817647241577480212L;
```

Figure 3: The Gaia Source Classification project viewed in Eclipse.

Figure 3 shows part of the Eclipse window with the Gaia source classification project loaded. The *Package Explorer* tab on the left side of the page shows the organisation of the project Java code into packages that do different things. There are six packages in total that contain Java classes that have the following purposes:

- *algo* contains the definition of algorithms used within the project.
- *algoimpl* contains implementations of algorithms used within the project
- *dm* contains 'data model' classes, e.g. the `Window` and `Source`
- *exec* contains executable classes, e.g. the main `Window` and `Source` processing applications
- *infra* contains infrastructure classes used by the source extraction and classification GUI
- *util* contains utility classes used for various purposes such as file loading and saving

Only three of the packages are important. The classes in the *exec* package are used to load and process files containing `Window` and `Source` data; these will be required when developing and testing source classification methods. The *algo* package contains Java classes that specify the interfaces for the source detection and classification algorithms. Have a look at the `algo.SourceClassifier.java` class: this is a Java *interface* that specifies the methods that any new source classification algorithms must implement. By enforcing source classification algorithms to adhere to this template, it means that we can easily integrate new algorithms into the project. The *algoimpl* package contains implementations of the algorithms specified in the *algo* package. These are the algorithms that do the actual work of detecting and classifying sources.

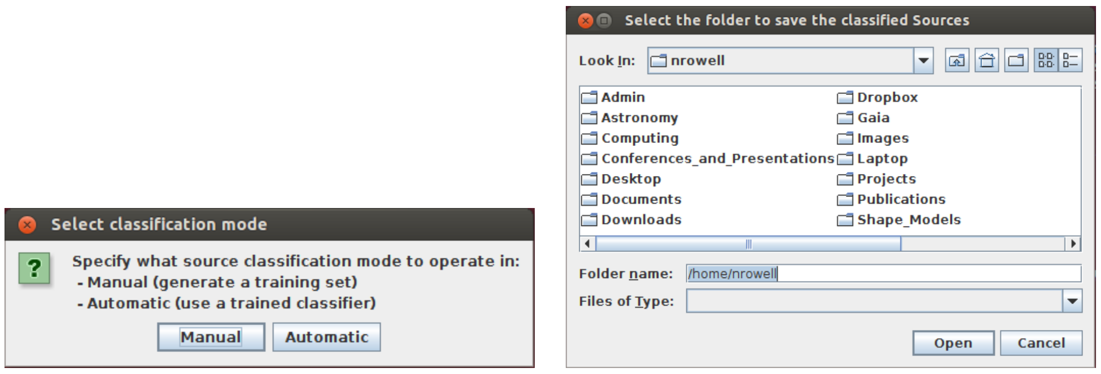
In terms of actual coding, the main task of this project will be to write a new source classifier algorithm that implements the SourceClassifier interface.

3 Source extraction and classification application

A special software application has been written for this project, to help with visualising and processing the Window and Source data, testing different classification algorithms and creating a 'training set' of Source classifications for use in developing machine learning methods.

3.1 Launching the application

To launch the application from within Eclipse, navigate to the *exec* package, right click on `GaiaSourceClassificationApplication.java` and hit Run As > Java Application. You will be prompted to choose either Manual or Automatic classification mode; Manual mode is for when we want to provide the Source classifications manually in order to generate a training set of data, and Automatic mode is for when we want to test an automatic classification algorithm. If you select Manual mode you will also be prompted for a directory to store the training set file. These windows are displayed in figure 4 below.


Select classification mode

Specify what source classification mode to operate in:
- Manual (generate a training set)
- Automatic (use a trained classifier)

Select the folder to save the classified Sources

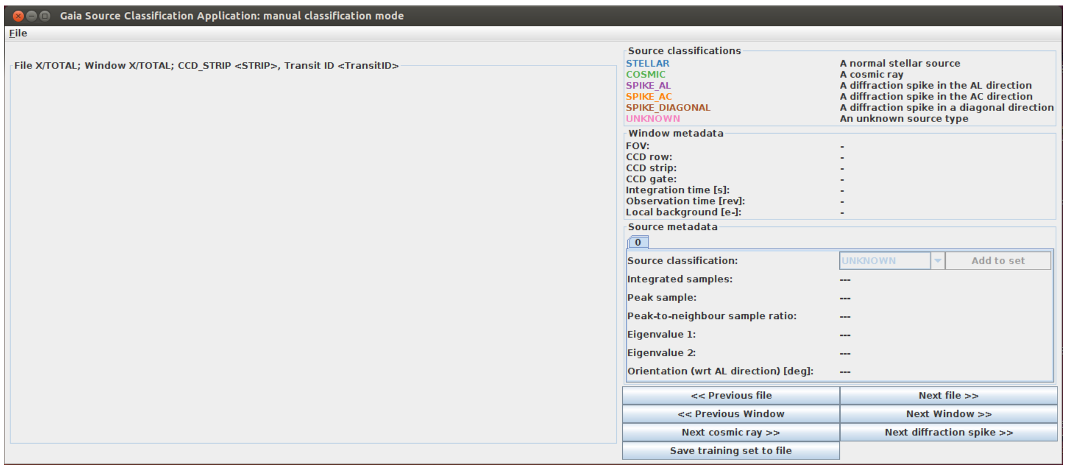
Look In: nrowell

Folder name: /home/nrowell

Files of Type:

Figure 4: The Gaia Source Extraction and Classification Application introductory windows.

Once the application has launched, you will be presented with the main GUI window as shown in figure 5


Gaia Source Classification Application: manual classification mode
File
<< Previous file
Next file >>
<< Previous Window
Next Window >>
Next cosmic ray >>
Next diffraction spike >>
Save training set to file
Add to set

File X/TOTAL; Window X/TOTAL; CCD_STRIP <STRIP>, Transit ID <TransitID>

Source classifications

STELLAR	A normal stellar source
COSMIC	A cosmic ray
SPIKE_AL	A diffraction spike in the AL direction
SPIKE_AC	A diffraction spike in the AC direction
SPIKE_DIAGONAL	A diffraction spike in a diagonal direction
UNKNOWN	An unknown source type

Window metadata

FOV:	-
CCD row:	-
CCD strip:	-
CCD gate:	-
Integration time [s]:	-
Observation time [rev]:	-
Local background [e-]:	-

Source metadata

0

Source classification:	UNKNOWN
Integrated samples:	---
Peak sample:	---
Peak-to-neighbour sample ratio:	---
Eigenvalue 1:	---
Eigenvalue 2:	---
Orientation (wrt AL direction) [deg]:	---

Figure 5: The Gaia Source Extraction and Classification Application main window.

The final thing to do is to load some data. This application processes files containing `Windows` as explained in section 1.2; these will be supplied as part of the project material. Click the 'File' menu at top left then select 'Load Windows', and enter a directory containing files of `Windows`. They will then be loaded into the application.

3.2 Automatic mode: testing a source classification algorithm

In Automatic mode, a built in source classification algorithm is used to classify each source as a particular type. This is useful for testing out new source classification algorithms.

3.3 Manual mode: creating a training set of classifications

In Manual mode, the source classifications are provided by the user via a drop-down menu for each extracted source. The user can then add each source to a training set by clicking the 'Add to Set' button next to the source type menu. The complete training set can then be saved to disk for later use in training a machine learning algorithm. In this mode, a secondary window is presented which shows the numbers of each type of source that have been added to the training set, and provides a button to use to save the training set contents to a file.

Because the number of stellar sources greatly outnumbers cosmic rays and diffraction spikes, it's impractical to build a training set by searching through all the objects. Therefore an initial source classification algorithm is included to roughly filter the sources, so that the user can easily search for sources of each type to add to the training set.