Geha Research Writeup

Chunyang Ding

July 2016

1 Project Summary - Coding

Classifying astronomical spectra is widely used to determine characteristics of the object in study. Even basic properties such as the presence of emission and absorption lines can be beneficial for the understanding of the morphology of a galaxy. The analysis of these characteristics can be done through Python, a common programming language for scientific data analysis. Surprisingly, analysis of spectral lines is not implemented in AstroPy, NumPy, or SciPy. In order to solve this problem, I developed my own code for the identification, fitting, and subsequent classification of spectral lines, with the goal to classify galaxy spectra as either emission or absorption.

The data used in the testing of the code comes from the SAGA collaboration, which has the goal of understanding Satellites around Galactic Analogues. They have collected many nights of galaxy spectra from the Multiple Mirror Telescope at the Whipple Observatory in Arizona. The data is in .FITS format with wavelength, flux, and inverse variance data. There is a corresponding .zLog that contains a calculated redshift and a manually assigned quality factor for each spectrum. The wavelength data is between 3500 and 9300 angstroms. In addition, additional data has been collected from the Anglo-Australian Telescope in New South Wales

The simple premise of the code is to take some FITS file and classify each of the spectra as one of four categories: Emission, Absorption, Bad Quality, or Needs Review. In order to do so, the code identifies the calculated locations of spectra lines, searches in the spectra for some present peaks or dips, and then fits the peak to a Gaussian profile. If such a regression is successful, it uses the parameters of the Gaussian fit to classify the spectrum. There are built in functions to automatically classify an entire .FITS file, as well as a debug utility that writes each of the parameter fits along the way, including plots of fitted regions.

The basic functionality of the code is to run the command

```
allObsFits = allTogetherNow(FITS_FileName, zLog_FileName)
```

to get the Gaussian fit data for each peak and then

```
print allClassified2(allObsFits)
```

to get the automatic classification. This classification uses the following keywords:

- -5: Needs Review
- 1: Bad Quality
- 5: Emission Spectra
- 6: Absorption Spectra

In order to run this code, you need to define two functions to process the data properly. You need to ensure that there is an appropriate .zlog file, which contains information regarding the quality of the spectra, and to have data in the form of wavelength, flux, and sigma. This is advised to be written as `def AAADataImport(fits_file, zlog_file)` and `def AAADataClean(all_data, fileNum)`. Afterwards, these functions ought to be called in the main `allTogetherNow( fits_file, zlog_file)` function.

In order to write to file the diagnostics information, simply run the command

`DiagnosticsFits(FileNum, ObjNum)`

and the appropriate graphs and data files will be written to the same directory as the location of the code.

The methodology of the regression code is multistepped, with several verifications of good data. It uses a nonlinear least squares fit for the following Gaussian form:

$$\frac{A}{\sigma \cdot \sqrt{2 \cdot \pi}} \cdot e^{-\frac{x-\mu^2}{2 \cdot \sigma^2}} + B$$

Where A, the amplitude of the Gaussian fit, can be positive or negative, but σ, the standard deviation, is only allowed to be positive. This implies that the sign on A determines whether the peak is a emission peak or an absorption dip.

First, the code verifies if the spectrum is a good quality spectrum, determined as Q ¿ 2. Qualities of 1 or 2 are assigned to sky spectrums as well as noisy, low signal to noise ratio (SNR) spectrums. Low quality leads to an automatic classification of 1. In order to classify a good spectra, several different spectra lines are selected for and subsequently tested against the calculated redshift, according to the redshift formula

$$\lambda = z \cdot (1 + \lambda_0)$$

If the shifted spectral line is within the range of 4000 and 8500 angstroms, the code assumes that this line could possibly be found. It selects a 30 angstrom window around the predicted line location to search for either peaks or dips in the flux data. A peak or dip is determined when the flux of a data point is greater than three sigma from the average flux of the nearby 30 angstrom region. If no significant deviation is found, an error value is returned to the main body of the code. Otherwise, the location of that peak/dip is used in further selection.

Subsequently, a 18 angstrom window is selected around the peak in order for regression. The regression is two fold, using the `scipy.optimize.curve_fit` routine, which takes as input the data region as well as the function to be fitted to. In order to optimize the fit, the code first fits to a normalized Gaussian form of the data, where the mean wavelength is shifted to zero and the flux baseline is shifted to zero. After receiving the parameters for such a fit, the original data is used, with the best-estimate parameters of the test fit as the starting guess for the true fit. This ensures a faster convergence of the fit with fewer errors.

If no fit can be found with the `optimize.curve_fit` routine, an error message is returned. Otherwise, the best fit parameters are returned along with the covariance matrix resulting from the fit.

Using this fit, two additional pieces of information is computed. First, the reduced Chi Squared test statistic is computed, with the formula

$$X^2 = \sum \frac{(o_i - e_i)^2}{\sigma_i^2} \cdot \frac{1}{\text{d.o.f}}$$

where the degrees of freedom is defined as the number of points used less four, to account for the four parameters that are fitted.

Next, the Confidence Intervals are calculated for the parameter fits. The code automatically calculates the 95% (2 sigma) confidence interval, although this parameter can be fine tuned by the user. This uses the t-statistic for the given confidence level as well as the results of the covariance matrix outputted by optimize.curve_fit. The confidence interval for parameter θ_0 is

$$\theta_i \pm t_{1-\frac{\alpha}{2},\text{dof}} \cdot \sqrt{Covar_i}$$

where $Covar_i$ stands for the diagonal entry in the covariance matrix corresponding to the ith parameter.

Given all of the regression information, the code is prepared to make an initial classification. It first looks for "good" lines, which are defined by the confidence interval for parameter A: if the 95% confidence interval is strictly positive or negative, and the sigma to A ratio is near or less than 1. From here, the code enters a logic structure based on our astrophysical understanding of the star. Please see Figure 1 for an examination of the logic structure that the classification algorithm uses.

The code can also calculate the sum and integral of the fits. While this can make a very accurate fitting algorithm, it can be easily influenced simply by the arbitrary flux units of the telescope and the choice of number of lines to include. Therefore, it is strongly advised to not use this code function (as `allclassified((allObsFits)`) but to use the updated classification scheme.

Examination of this code still reveals some bugs, especially in a miscalssification of absorption line spectra as "needs review" spectra. This is especially present in weak absorption line spectra, where there may be very faint emission line spectra in the Hydrogen-alpha regions. Unfortunately, this is an outstanding bug. Fortunately, these are simply marked as "needs review", so any scientist can go through these casewise and quickly classify those unknown spectra.

In addition, the code does not currently account for QSOs, or Quasar Objects. This is primarily due to the difficulties involved in dynamically selecting a data window to perform the regression across. As quasar lines are very different in shape and size, it is difficult to predict exactly what data should be

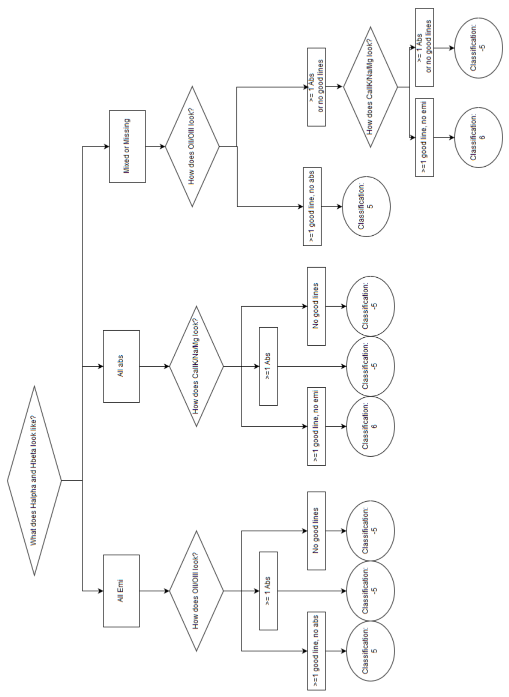

Figure 1: Flowchart for logic of the code.

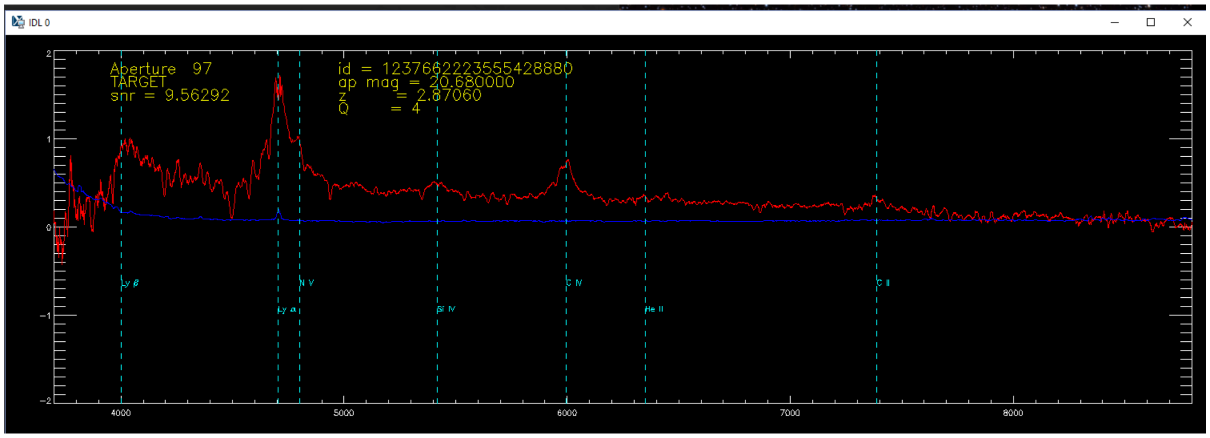

Figure 2: A sample QSO spectra from the data set

selected. Although it is obvious to the human eye, it is much tricker to predict with the computer algorithm. Although the peak of the data could be found, the window size was very off and resulted in poor fits that could not be used for any predictive capability. Therefore, through this project, we decided to instead go through and identify each of the QSOs by eye for possible use in other astronomical projects.

2 Classification of Objects

In order to train the code and to test if it worked, the current data set was entirely classified by eye. This included 15,900 objects from the MMT and roughly 5670 objects from the AAT. This data is available to be used with permission from Professor Geha. For each object, a classification was assigned by eye and additional notes were put on the side.

In addition to classification of these objects, objects that appeared to have a bad quality marker were flagged to be reviewed by the PI. A suggestion was made to change the quality of those objects, based on the presence and strength of different lines, the spacing between lines, the shape of the spectra, and the noise in the spectra. 274 spectra were advised to be reviewed; more than 200 of them included an upgrade in quality. This helps expand the data set used for future machine learning training, especially as the spectra here tend to represent edge cases of quality.

Finally, Quasar objects were identified by eye and flagged separately. 68 objects total were identified as Quasar objects, and their id and spectra were recorded. This could be useful for scientists who study active galactic nuclei, as it provides additional data to be used. See Figure **??** for a sample QSO spectra.