Identifying Young Stellar Objects

Ishaan Bhojwani, Alexander Tyan, Kevin Sun, Tyler Amos

Background

One of the main areas of research in astronomy is in stellar life cycles: the study of how stars evolve over time. Astronomers involved in this field are often hard pressed to find interesting objects to look at. For example, while a star may stay in the main sequence (the middle portion of its life) for billions of years, it only takes a fraction of that time (on the scale of several million years) for a star to develop or die. This means that the vast majority of stellar objects are main sequence. While data about these stars is certainly useful, main sequence stars are fairly well understood, whereas the birth and death of a star is more mysterious. Many astronomers are then faced with a problem: if main sequence stars are orders of magnitude more common than other stellar objects, then how do they practically find young or old stars to study?

One potential solution for finding young stars comes from the unique state in which Young Stellar Objects (YSO's) are normally found. Stars form when large clouds of gas coalesce, meaning YSO's are often surrounded by gas and dust. This dust absorbs portions of the light emitted by the star, becoming energized. As the dust loses this extra energy, it emits the absorbed light as lower energy photons. The overall effect this has is that the star will appear to emit much more low energy (infrared) light than expected of a regular star (infrared excess, or IRXS). However being a color outlier alone does not directly imply an object is a YSO, as several other objects display similar color characteristics such as quasars and colliding galaxies. However, YSO's are unique in that they mostly form in clusters, as gas clouds large enough for star formation usually contain more than enough gas for several stars (if not thousands). This creates the possibility of searching for YSO's by identifying clusters of color outliers.[1]

Objectives

Our main objective was to create a method for identifying YSO candidates from a dataset far too large for manual analysis. In practice, such a method could be used by YSO researchers to quickly identify interesting candidates for follow up research.

Dataset

We chose to use the ALLWISE source catalog. NASA's Wide-field Infrared Survey Explorer (WISE) mission took mosaics of the entire sky in 4 infrared bands (3.4, 4.6, 12, and 22 microns). The initial WISE mission provided 42 billion time-tagged measurements containing a variety of photometry and position data. These images were then overlapped where applicable, resulting in the final ALLWISE data release of 747 million measurements. We chose to use the ALLWISE release instead of the original WISE dataset as the ALLWISE release filtered for rejects and bad readings. Furthermore, ALLWISE turned WISE time series data into proper motion estimations. We estimated the entire dataset to be around 815 GB. We ran our analysis on about 13 million objects (13 GB) of this larger dataset, as we filtered only for objects with motion estimations and with a signal to noise ratio (SNR) greater than 2 in all channels. This SNR cut off, while greatly limiting the size of our data, improves the likelihood that a given object exists, rather than being a sensor artifact. Lowering this threshold to SNR >= 0.1, for

[1] Struck-Marcell and Tinsley

example, would have increased the size of our dataset to dozens of gigabytes, however as our analysis was fairly intensive to begin with we chose a higher SNR for more precise results. Similar studies done by hand have placed their cutoffs at SNR's of 5 or 10.[2]

Method

We created two separate algorithms to fulfill the above mentioned goal. Common to both algorithms was searching for color outliers and clusters. IRXS analysis was done through the use of color-color comparison, where the ratios of flux at different wavelengths are plotted against each other. We used a standard approach of comparing:

$$w1 - w2 \text{ vs } w3 - w4$$

with the micron values of each band being:

$$w1 = 3.4,\ w2 = 4.6,\ w3 = 12,\ w4 = 22$$

Note that magnitudes are logarithmic, so subtracting magnitudes is equivalent to calculating the ratios of flux. Outliers were those objects with irregular color-color ratios. This is a standard approach for IRXS analysis.[3][4] Both methods were implemented in MapReduce, although early versions of our k-means algorithm were done in Spark.

Also common to both algorithms was how we defined distance between objects, although our methods for turning this distance into likely clusters were very different. We had accurate and precise right ascension (ra) and declination (dec) readings, placing every object at a specific point in the sky, so comparison in sky position was not difficult. However, without redshift measurements it is impossible to judge the distance from earth of each object. In lieu, we used the proper motion estimations provided by ALLWISE. We calculated the distance between objects in a 4D space consisting of the ra, dec, and motion in ra/dec of the object. Attempting to judge 3D distance from a 2D projection required us to include some other metric besides their position in the sky

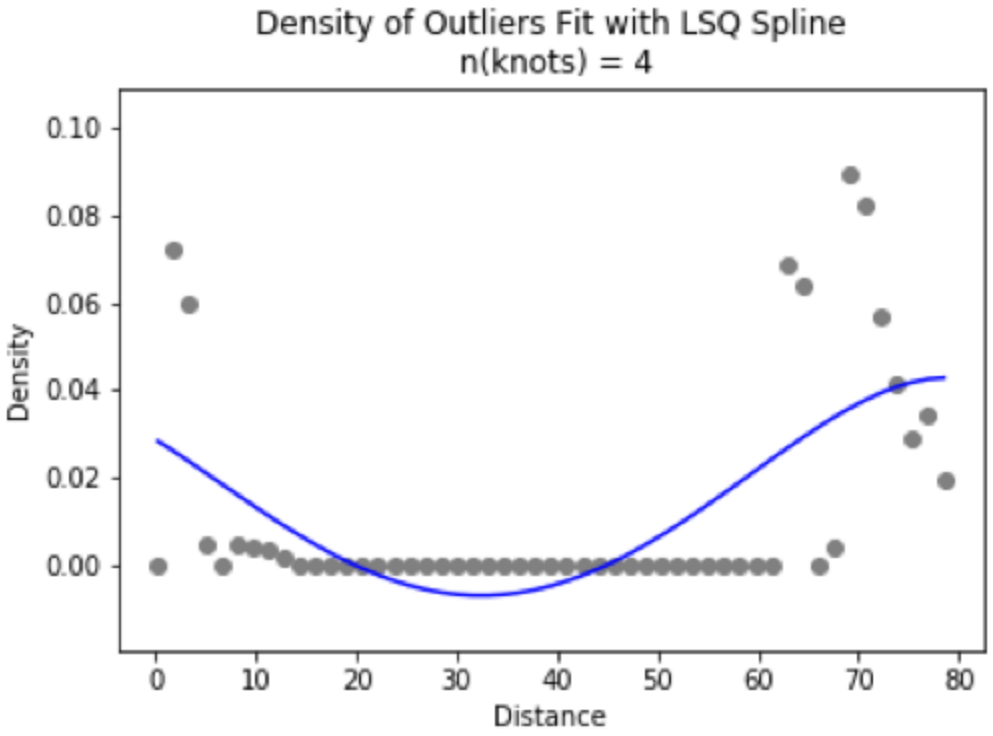
Algorithm I

Preliminary analysis showed a distribution of color-color ratios centered around two points, (0.063, 0.925) and (0.37, 2.92). Around these centers was a distribution which we decided to treat as normal. To find color outliers, we implemented a k-means algorithm in MapReduce and grouped all objects into one of the two groups. We then calculated the standard deviation within each group from each centroid. Any object with a color further than a set multiple of the standard deviation would then be treated as an outlier. As the distribution was far from truly normal, we used a fairly strict cutoff.

After we had identified color outliers, we used a pairing technique to draw edges between outliers. At each node, we maintained a running sample of nodes against which we calculated edge distances. We then selected the top k closest nodes and fit a quadratic spline over the distribution of those points' density. This method was chosen based on heuristics and observation of data in the process of development. The first saddle point, a local minimum, was derived from the spline and returned as the boundary of a given cluster.

---

[2] Gorijan et al.
[3] ibid.
[4] Struck-Marcell and Tinsley



Algorithm II

While we were fairly confident in our k-means approach for getting color outliers, we wanted to try a different clustering mechanism. Our second algorithm approaches clustering in a very different way. We partitioned the sky into boxes and ran a random walk and thresholding algorithm within each box. The random walk involved calculating the distances between every object within the box, and then turning that distance into a jump probability, where a smaller distance meant higher probability. A single run of the random walk involved starting at a random object within the box and making *n* jumps with the destination object being decided by a random draw based on the distance-probability previously calculated. We recorded the total number of times each object was visited across many random walks. The number of visitations then correlated with how 'central' an object was, that is, how densely packed its region of the box was. Stars in areas with sparse distributions were visited less often. We then subdivided this box and counted the total number of visitations for all objects within each subcell. Our theory was that the subcell with the highest total visitation count contained clusters. Using a skimage thresholding algorithm, we found the 'foreground' subcells and returned all objects within them as objects within clusters.

As this method was reliant on physics density of objects, we filtered for color outliers only after clustering. To do so, we used the same k-means and standard deviation technique as algorithm 1. As previously mentioned, this was entirely done in MapReduce, although early versions of k-means were done in Spark but moved to MapReduce to unify our codebase and so that we could write our own implementation.

Challenges

We faced numerous challenges during this project, time constraints aside (see Addendum). One of the most significant compromises we had to make was how we defined color outliers. The color-color data was far from normally distributed to begin with. For simplicity as well as the ability to continue under limited time, we decided to treat the data as bimodal. This is an uncertain assumption on 2 levels: first, it does appear that there are two centroids for the color-color data, but one of the groups contains much more of the data than the other. Within the larger group there may have been a third centroid we did not pick up on in our preliminary analysis. While there is some scientific justification for there being at least two centroids (galactic vs extragalactic objects expressing themselves different), we made this decision by examining graphs of color-color plots from a subset of our data and through some light kmeans experimentation with 3 or more centroids. To be more precise would likely require deeper machine learning analysis. Secondly, we assume that around these centroids the objects are normally distributed. This is clearly not the case, hence how strict we were with our standard deviation cutoff. However, without a time consuming machine learning analysis or a manual demarcation of outlier bounds, we could not think of a way around this issue. The strict standard deviation cutoff we had to enforce likely left us with a significant false negative rate However we feel confident that our false positive rate is low. For compiling a list of objects for YSO follow up research, having a low false positive rate is far more important than a low false negative rate, so we do not see this issue as invalidating our results in any way.

There is also the question of parameters for several of our functions. For example, we do not know how many bins we should partition the sky into, how many steps should be in each random walk, or how many random walks we should perform to get the best results possible. The parameters we used were from ballpark estimates about the density of stars in a given region and from how fast our algorithms would converge. In order to solve this, we would need to have a way to quantify how ‘good’ our current parameters were. For clustering, this could be be confirmed through a lengthy checking process, where we check our potential clusters against the original data again and see how likely the objects are to be related. However, for color outliers it is impossible to confirm the identities without full spectra of the objects. Spectra are rare, as they require special equipment, and are usually only performed on a per-need basis, which means only the smallest fraction of our objects have spectra available.

Third, the best clustering technique remains an open question. Clustering 3D objects based on their 2D projections is difficult, however we used several tools to make this easier. Most prominently, we factored in the estimated movement of the objects in our distance calculations. This meant that objects were considered ‘closer’ if they were moving in the same direction across the sky, and at the same speed. However, we do not know the best way to weight speed versus position. If two objects are very close to each other, yet moving in opposite directions, are they separate? Or is it a binary YSO system, with the two objects orbiting each other? The first case is meaningless, the second is potentially very interesting. One way to help solve this could be to implement magnitude in our ‘distance’ calculations. Objects with different magnitudes could have more ‘distance’ between them. Of course, this is not perfect, as stars of different magnitudes commonly appear in binary star systems.

We also faced several technical challenges related to the Dataproc execution of our two algorithms. While we could successfully execute our code locally on smaller data sets (up to

around 365,000 observations), testing and debugging on Dataproc turned out to be difficult. One of the reasons is that mrjob and Dataproc error messages are not very interpretable. Without a traceback Python, it was difficult to pinpoint at what stage in our multi-step jobs the issue occured. Since the setup times of running an algorithm were high, testing by excluding small chunks of code was difficult under time constraints.

Another reason is what we suspect to be Dataproc's difficulty in executing MRJob tasks that involved the use of our helper functions and classes external to the MRJob code. The code seemed to make it through more steps when we moved more code into a single .py file for each algorithm. In the end, we were unable to fully rewrite our algorithms to execute them on a Dataproc from start to finish. Thus, we decided the best we could do was splitting a subset of the data and running our algorithms across three machines locally to simulate computational scale-up (see results).

The bulk of our challenge with Google Cloud's Dataproc environment was related to resolving environments and successfully installing required packages on the remote master and workers. Specifically, our implementation of Algorithm 2 required functions from sci-kit-image. However the most recent implementation of these functions differs substantially from the version which is installed by the Python package management system pip. Moreover, there is a persistent issue with a conflict in dependencies of sci-kit-image's dependencies, more specifically the requests module. While this issue appears to be fairly common,[5] a solution which could be specified via shell script, uploaded to the cluster, and run without user intervention was not readily apparent. Strategies attempted included: installing packages from a requirements file based on a successful local environment; installing anaconda distributions with pre-packaged distributions of the modules we needed; cloning repositories and building from source; installing and removing packages via bootstrap shell scripts (see bootstrap.sh lines 13 - 15), and using deprecated or alternative functions. One successful example of this last strategy was using "float('inf')" in lieu of the inf. function in the math module.

Finally, we ran into quota issues with Google Cloud computational allowances. We had to request increases for hard drive space, CPU cores, and in-use IP addresses on multiple occasions and only after we found tasks to be too demanding for previously allowed resources. Sometimes, the approval process took several days (some requests are still processing). In combination with Dataproc execution debugging and restarting the code on Dataproc, we ran out of the billing resources on an account which had the requisite expanded allocations to run our first algorithm.

Lessons Learned

One of the important lessons learned was the development workflow for Big Data projects such as this one. We learned that running MRJob tasks locally and on the cloud can be quite different, not only in terms of set-up costs, but also the design of the code itself. As we mention, it seems recommendable to integrate as much of the code as possible into a single MRJob class, including job-necessary variables in the "_init" function definitions. Had we known that there would be such drastic design difference necessary between the local and Dataproc versions of the code, we could have designed a more incremental approach. Assuming that the code that runs locally does not necessarily run on Dataproc, we would have added more

[5] See, for example:
https://stackoverflow.com/questions/27341064/how-do-i-fix-importerror-cannot-import-name-incompleteread

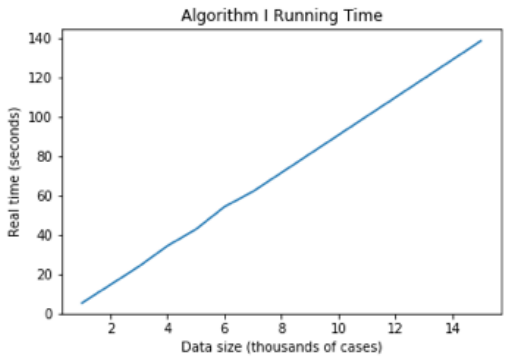
Dataproc testing at critical decision points in the coding process. For instance, this could be done for testing the splitting of helper functions away from the MRJob code or the integration of the AstroObject class into our code.

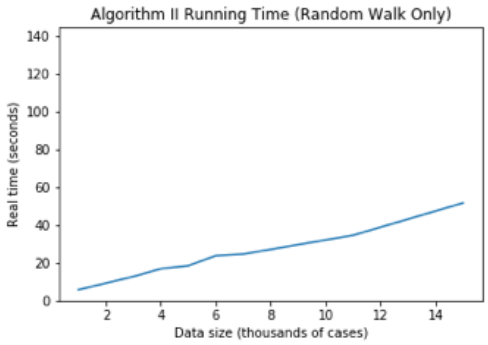
Dataproc implementation aside, we learned that MRJob objects can inherit each other, as well as use custom-made class objects. This allowed for a more readable and clean code. The implementation of our AstroObject class also forced us to learn more about MRJob's handling of programmer-defined objects. It turned out that using Pickle protocols for passing the class object data between sub-steps of a job is more reliable than the default JSONProtocol. Pickle can parse attributes of a programmer-defined class object and convert to string representations and vice-versa. This made it possible to pass object information within each step. Likewise, we chose TextValueProtocol for the final output of information over MRJob's default. This allowed for human-readable CSV lines output, each line representing an AstroObject, with attributes that may of interest to a researcher.

We also learned that, even with access to on-demand computational power, one must still keep in mind the scale of computation when designing the algorithm. For instance, we suspect that one of the reasons our Algorithm I could not finish on a Dataproc cluster was because the search process spawned a large byproduct of data that needed to be stored on cluster machines along the code execution. Once the space limits were reached, the code could fail. This was hard to notice on small datasets with local testing. In hindsight, one way we could have improved computational efficiency of Algorithm I was implement a similar binning technique that was used in Algorithm II. This would mean that the edge drawing would happen only between color outliers within the same group. Such an implementation could drastically reduce computational requirements given the size of our data set.

Runtime Summary and Justification for "Big Data" Methods

Algorithm 1, running locally on a modern laptop, had a runtime in excess of 85 minutes for a CSV with 365,000 rows. By comparison, an early version of Algorithm 2, running locally on the same laptop with a dataset of 365,000 rows, finished in approximately 16 minutes.





For a dataset with a total size of approximately 800,000,000 rows, a local or serial implementation was thus infeasible. Even with the reduced dataset with which we worked (approximately 15 million cases), these runtimes would be prohibitive. It is clear from the above

plots that the second algorithm is significantly more efficient, although both algorithms appear to scale linearly.

Moreover, the first algorithm revealed itself to be quite error prone, especially with respect to memory. While the group was not able to pinpoint the exact bottleneck in the first algorithm, a scaled-up parallel implementation of the first algorithm resulted in persistent memory errors, as the MapReduce framework exceeded the nodes' available memory. When troubleshooting locally on a modern laptop, a substantial dataset could quickly use all the laptop's available memory through its creation of temporary files and provoke a memory error.

Summary of Results

Algorithm	Success Metric	Degree of Success
Algorithm 1	Comparison of sample of results to WISE images	Based on visual confirmation, it appears the first algorithm successfully identifies interesting objects.[6]
Algorithm 2	Comparison of results to larger WISE imagery	The second algorithm successfully generates a visual "map" of the sky as a heatmap. This would enable a researcher to identify objects which "co-exist" and are thus likely to form clusters in a given stellar region.

In total, our successful implementations identified over 65,000 objects in 358,000 cases and 103,000 objects in 15,000,000 cases, respectively. The latter figure is more authoritative due to the more robust performance of the second algorithm, in terms of both runtime and robustness to the constraints of the machine (i.e., memory). This latter detection rate corresponds to approximately 0.06 %, which may indicate this algorithm is indeed identifying unique or unusual groups of objects.

Conclusion

It appears to be possible to identify young stellar objects from multispectral data using distributed computing environments. Adapting existing algorithms proves useful in flagging outliers and identifying unusual objects which appear to be clustered with other outliers. Applying standard graph algorithms such as random walks, and image thresholding methods from computer vision we are also able to generate a less granular but more comprehensive, and less fault-prone picture of where the most promising objects lie.

[6] See images "J002553675826013_colour_algo1_result.png", "J003547983153528_colour_algo1_result.png", "J003556512948146_colour_algo1_result.png" in "presentation" folder of the repository

We believe these outputs provide a rough guideline for where follow-up astronomical investigation would be most fruitful. Future investigations of this topic could leverage labelled training sets and forms of machine learning which are amenable to distributed implementations.

Addendum: Impact of Changing Topic

The group identified at approximately week 8 of the 10-week quarter that their existing project, the code for which can be found in this project's GitHub repository, was unlikely to be completed in time for project deadline. As such, the team began anew with a new topic relatively late in the quarter. This meant that some of the algorithms and implementations were still in the process of maturing as the quarter came to a close. In this two week period, the team identified a new research question, identified and collected data, developed a research plan, and implemented multiple iterations of algorithms which contained substantial original content.