Efficiently Searching for Stellar Flares in Kepler Mission Data

Nicole Loncke & Lucianne Walkowicz

May 18, 2014

1 Introduction

The Kepler Mission is designed to survey our galaxy in the hopes of discovering planets in or near the habitable zone of their stars. In doing so, it will become possible to determine how many of the billions of stars in our galaxy have such planets. Kepler stares at a small portion of the sky for years at a time to gather brightness data about these stars. In addition to planet detection, this data can be used to gather properties about the stars themselves. In this paper we are concerned with the flaring behaviors of these nearby stars and discovering techniques we can use to better determine their effect on the environments of any orbiting planets.

2 The Data and Motivation

The data we chose to investigate are from Quarter 9 of Kepler's collection. This corresponds to the 3-month period from mid-March to late June 2011. Within that time window, we examined M dwarf stars in particular. M dwarfs comprise about 75% of the stars in our galaxy so they are primary candidates for investigation, if only due to their widespread representation in our neighborhood and in the universe. Furthermore, M dwarfs are highly magnetically active—when magnetic field lines snap and reconnect, the resulting outpourings of energy can have drastic effects on the conditions of nearby planets.[1]

Our over-arching goal was to collect measurements on the frequency and intensity of flares for each light curve. The process is two-fold: first, one must correctly identify the flares in the data; secondly, one must compute the frequencies and intensities, then analyze the results. In the following section, we discuss the code we wrote to assist in performing these tasks.

[1] Hilton et al., "The Galactic M Dwarf Flare Rate," 2010.

3 Our Tools

One rudimentary method of finding the flares is to flag those points in the light curve that have brightness above a certain threshold. We have written a program in IDL that does this and must verify by-eye all of the events it flags as flares. To do so, we have also written a Python module, `lightcurves.py`, to aid in the vetting process.

3.1 Formatting

The `lightcurves` module makes some assumptions about the format of the input data. The light curve data should be in a file containing a whitespace-separated table with time in the first column and flux in the second.

808.51470	*6338.22*
808.53514	*6340.73*
808.55557	*6346.89*
808.57601	*6341.10*
808.59644	*6340.22*

Table 1: Light curve data sampled from Kepler ID 10068383.

The other fundamental input file is the one containing the potential flare events, or "flags," generated by the IDL program. This file contains one column listing indices into the time array at the points that mark a suspected event. The module functions use the light curve data and flags thusly formatted as inputs for vetting.

3.2 Plotting

The `ltcurve()` function takes as its primary argument a string of the name of the file containing the Kepler data and returns the time and brightness arrays. By default it also displays the light curve corresponding to the file on a time vs. brightness plot, but this feature can be switched off by passing the function an optional argument.

The `ltcurves()` function displays multiple light curve files one at a time. Its only required argument is a list or array of filename strings. This function also accepts a keyword argument `flags`—a list of corresponding event flags for each of the Kepler data files—with which it overplots the potential flares.[2]

3.3 Vetting

Instead of cycling through the light curves with overplotted flags, you may find it helpful to inspect and record whether or not the marked events could potentially be stellar flares. In

[2]These flags are generated using `getflags()` by passing it a list of the names of the files holding the flare flags.

that case, you should use `flareshow()`, which writes user input (either 'y', 'n', 'm') to two files for later retrieval. See examples of the display in figures 1, 2, and 3.

Figure 1: A true stellar flare, displayed with flareshow().

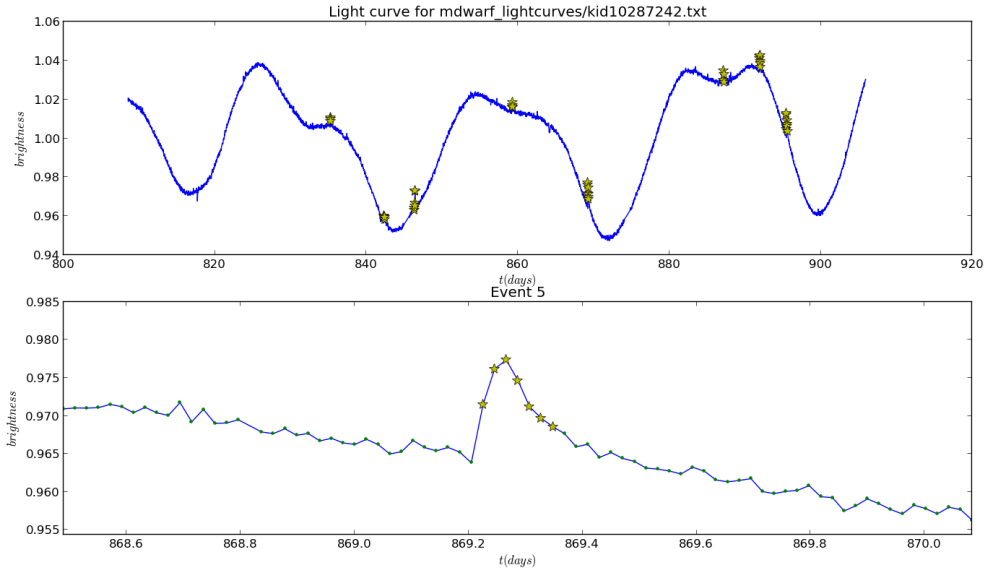
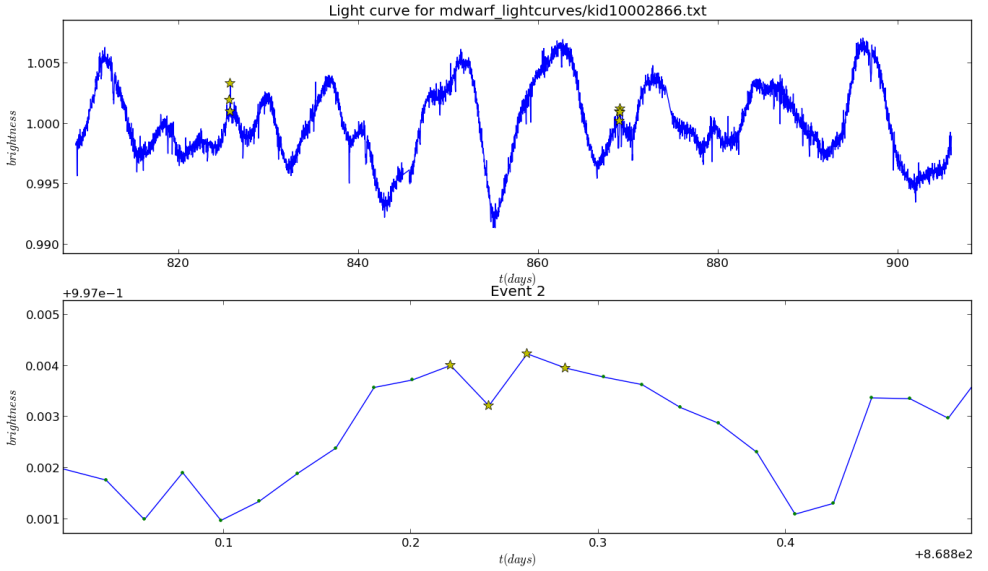
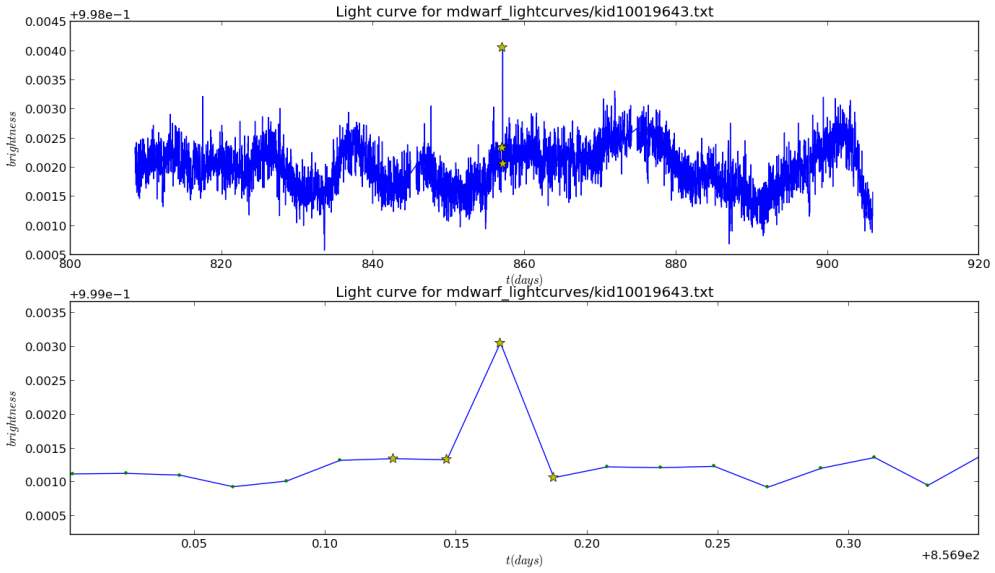


Figure 2: A falsely flagged event.



One file contains a space-separated table of the Kepler IDs and the corresponding user

Figure 3: An indeterminate event.



responses to its events, see Table 2. The other file contains information about the length of each event, as displayed in Table 3. These two files work in conjunction to gather more information about the potential flares.

8848271	n
8908102	n
8953257	n n n n n n n
9002237	n n n y

Table 2: Example output.txt file.

8848271	3735 03
8908102	1757 03
8953257	1454 6 1610 7 1890 4 2359 3 2516 4 2829 5 2985 6 3265 5
9002237	3337 4 3547 5 3756 3 3967 4

Table 3: Corresponding example output_indices.txt file.

Note that before using `flareshow()`, you must have your flags in the proper format, generated by `getflags()`. This helper function outputs a nested list of event indices given a list of the names of the files containing the flags.

4 Initial Approach

In order to gather data about the flares, it is necessary to first correctly identify the stellar flares within the Kepler data. These events have a very distinct shape and thus can be picked out from the light curve data by a simple program with rudimentary accuracy. We have written such a program in IDL that takes note of points in the light curve where the brightness exceeds a certain threshold. Before proceeding to collect data on the flares, we must first manually verify all of the events it flags as flares.

To make this verification process easier, we have also written a Python module, `lightcurves.py`. It contains functions that display the light curves with the flagged points marked, in addition to accepting and recording user input about whether the flags are truly flare events. We employed a combination of these tools to inspect over 300 events, {} of which we determined to be flares.

5 Machine Learning

This method of visually checking each event is slow, however, and the major bottleneck for data analysis. The IDL flare detection program that produces the flare flags recognizes some data metrics but does not correctly identify events with high accuracy. Our human brains allow us to do the same thing but with more nuance. You can see some examples for yourself in figures 1,2, and 3. If we could write another program to recognize the same patterns that humans so easily detect in the light curves then we could nearly entirely automate the data-gathering process—from the raw light curve to having information about flares with high confidence. To accomplish this goal, we trained a variety of classifiers on metrics from each potential flare event and their respective light curves.

5.1 Training

Our first task was to gather quantitative data about the stellar flares to feed into the classifier. In total we use 10 metrics.

(1) *amplitude*: the range of the entire light curve. Stars with great stellar variability tend to be more magnetically active than those without. We expect high light curve amplitude to correlate with real flares.

(2) *number of events*: Light curves that have many flagged events tend to have real flares, so we expect a high number of events to correlate with real flares.

(3) *standard deviation*: The standard deviation of the entire light curve with stellar variability subtracted. It may be useful to feed the classifier more information about the light curve at large.

(4) *consecutive points*: Sometimes there are gaps in the Kepler data. Kepler must rotate and point its antenna towards Earth to send its light curve data roughly every month. When the satellite begins recording again, there may be a sudden increase in brightness that resembles a flare but isn't. In order to avoid marking these as true flares we check whether the time intervals are evenly spaced across the event.

(5) *kurtosis*: The kurtosis measures the "peakedness" of a flare event. A sharp increase and decrease in brightness is likely to indicate a true flare, though the decay ought to be more gradual than the incline.

(6) *midpoint check*: A stellar flare typically requires a monotonic increase then monotonic decrease in brightness. Ensuring that the middle point is higher than the beginning and end points of the event is one way to rule out falsely marked events.

(7) *second derivative*: Smoothing over the flagged event, is the light curve locally concave up or down? The second derivative of the window around the potential flare can capture the shape of a light curve in the neighborhood of an event.

(8) *skew*: Skewness is a measure of the asymmetry of the event brightness—is the flare left-leaning or right-leaning? Because flares are characterized by very quick increases in brightness followed by a slow decay, left-leaning events (and therefore those with negative skew) are more likely to be true flares.

(9) *slope*: Is the brightness of the star generally increasing or decreasing at the time of the event? This metric measures the slope of the line formed by connecting the point at the beginning of the flare window to the point at the end of the flare window. time of the event?

(10) *slope ratio*: We also compute the ratio of the light curve's slope just before the event begins and the slope just after it ends. We hope to capture more information about the local shape of the light curve with this metric.

These data were gathered for 315 potential flaring events that we had previously labelled by-eye. Using these samples we formed a training set of 150 events and a test set of 165 events.

5.2 Classification Performance

We use Python's scikit-learn package for our machine learning framework. While it comes equipped with a suite of regression, clustering, and dimensionality-reduction tools, we are only concerned with classification. Our target classes are 'y' for definitely a flare, 'n' for not a flare, and 'm' for any indeterminate events.

	precision	recall	F_1-score	support
n	0.62	0.87	0.72	60
y	0.63	0.69	0.66	65
m	0.50	0.12	0.20	40
avg	0.60	0.62	0.57	165

Table 4: Linear kernel performance with the test set.

To quantitatively compare the classifiers, it is important that we define a few statistics related to their performance. We say the *precision*, or *efficiency*, is the fraction of events classified as a given type ('y', 'n', 'm') that are truly of that type. The *recall*, or *completeness* of the classifier is the fraction of objects that are truly of a given type that it classifies as that type. The F_1 score is a weighted average of recall and precision.

Though we ideally seek high scores for both precision and recall, for our purposes precision is the more important metric. Because we have many events in our dataset it is better to correctly identify a small number of flares than to find many true flares at the expense of false positives.

5.2.1 Support Vector Classification

For our first attempt we used support vector classification as packaged in `sklearn.svm.SVC`. We initially used a linear kernel SVM. This is a simple classification method which assumes that there is a hyperplane that separates the data in the feature space, which is 10-dimensional in our case. We trained our linear SVC on 150 flare events and then used it to predict the status of 165 events. This classifier had a precision of 63% for classifying true flare events. While the results were better than a coin toss, we sought to improve the classifier performance.

Sometimes the data cannot be linearly separated within the given feature-space. Alternate SVM kernels implicitly map the data to higher dimensions to find a separating hyperplane. To do that, we incorporated the popular radial basis function (RBF) kernel into our support vector machine model. While support vector classification with an RBF kernel does decently when predicting the flares it has already seen, it does not outperform when predicting unseen events, which is ultimately what is important. The results are charted in tables 5 and 6.

5.2.2 Random Forest Classifier

We performed the same task using random forest classification, as packaged in `sklearn.ensemble`. The algorithm constructs many decision trees based on the inputs it sees during the training phase, then uses those trees to predict the categories of unseen samples. While it performed superbly on the training set, it performed no better than our support vector machines on the

	precision	recall	F_1-score	support
n	0.73	0.81	0.77	57
y	0.74	0.89	0.81	61
m	0.71	0.31	0.43	32
avg	0.73	0.73	0.71	150

Table 5: Reconstructing the training set with RBF kernel.

	precision	recall	F_1-score	support
n	0.61	0.88	0.72	60
y	0.62	0.60	0.61	65
m	0.33	0.12	0.18	40
avg	0.55	0.59	0.55	165

Table 6: RBF kernel performance on the testing set.

	precision	recall	F_1-score	support
n	1.00	0.92	0.99	57
y	0.95	1.00	0.98	61
m	1.00	0.94	0.97	32
avg	0.98	0.98	0.98	150

Table 7: Reconstructing the training set with random forest classification method.

	precision	recall	F_1-score	support
n	0.63	0.77	0.69	60
y	0.60	0.72	0.66	65
m	0.29	0.10	0.15	40
avg	0.54	0.59	0.55	165

Table 8: Random forest method performance on the testing set.

test set.

5.2.3 Linear Discriminant Analysis

Lastly, we used the linear discriminant analysis method (LDA), which attempts to model the difference between the classes as linear combinations of the features. LDA is closely related to principal component analysis in that it can also be used for dimensionality reduction, but we employed LDA only for its capabilities as a linear classifier. Our LDA classifier performed with 60% precision and 71% completeness for the true flares, which is comparable to the other three methods.

	precision	recall	F_1-score	support
n	0.72	0.86	0.78	57
y	0.70	0.84	0.76	61
m	0.67	0.19	0.29	32
avg	0.70	0.71	0.67	150

Table 9: Reconstructing the training set with LDA.

	precision	recall	F_1-score	support
n	0.63	0.83	0.72	57
y	0.60	0.71	0.65	61
m	0.56	0.12	0.20	32
avg	0.60	0.61	0.57	150

Table 10: LDA performance on the testing set.

6 Conclusion

We set out to gather data about flaring stars in order to explore the impact that these stars might have on the conditions of nearby planets. Processing the light curves to identify flares, however, required a large human component in the form of looking at each curve and manually recording the status of each event. In an effort to automate flare detection, we employed machine learning techniques to classify a set of pre-vetted events. Of the four classifiers used—SVM with linear kernel, SVM with radial basis function kernel, random forest, and linear discriminant analysis—performance metrics suggest that the simple linear support vector machine is the best choice for our task, as it has the best precision for the test data.

There is still much room for performance improvement, however. As with many classification problems, increasing the size of the training set can significantly improve prediction accuracy. Doing so requires more labelled data which means we must, ironically, vet more potential flares by eye. As a diagnostic for expected improvement we can plot a learning curve (accuracy as a function of training set size) for each of the different classifiers. It may be that the classifiers have simply not seen enough variety in the training set to be able to classify new inputs. Alternatively, if the learning curves have already plateaued with a training set of 150 samples then providing more samples won't significantly improve prediction accuracy.

The advantage of being able to test the classifiers on labelled data is that we can see how many of the flares are being incorrectly identified. With this knowledge, there may be a way to correct for those flares that are misjudged by our algorithms.

Currently, we have yet to completely remove the human element from detecting light curves, but with the the machine learning techniques mentioned we have higher hopes for doing so.