The Astrophysical Journal, 820:138 (15pp), 2016 April 1 doi:10.3847/0004-637X/820/2/138

CLUSTERING-BASED FEATURE LEARNING ON VARIABLE STARS

Cristóbal Mackenzie[1], Karim Pichara[1,2,3], and Pavlos Protopapas[2,4]
[1] Computer Science Department, Pontificia Universidad Católica de Chile, Santiago, Chile
[2] Institute for Applied Computational Science, Harvard University, Cambridge, MA, USA
[3] Millennium Institute of Astrophysics, Chile
[4] Harvard-Smithsonian Center for Astrophysics, Cambridge, MA, USA
Received 2015 December 21; accepted 2016 February 11; published 2016 March 30

ABSTRACT

The success of automatic classification of variable stars depends strongly on the lightcurve representation. Usually, lightcurves are represented as a vector of many descriptors designed by astronomers called features. These descriptors are expensive in terms of computing, require substantial research effort to develop, and do not guarantee a good classification. Today, lightcurve representation is not entirely automatic; algorithms must be designed and manually tuned up for every survey. The amounts of data that will be generated in the future mean astronomers must develop scalable and automated analysis pipelines. In this work we present a feature learning algorithm designed for variable objects. Our method works by extracting a large number of lightcurve subsequences from a given set, which are then clustered to find common local patterns in the time series. Representatives of these common patterns are then used to transform lightcurves of a labeled set into a new representation that can be used to train a classifier. The proposed algorithm learns the features from both labeled and unlabeled lightcurves, overcoming the bias using only labeled data. We test our method on data sets from the Massive Compact Halo Object survey and the Optical Gravitational Lensing Experiment; the results show that our classification performance is as good as and in some cases better than the performance achieved using traditional statistical features, while the computational cost is significantly lower. With these promising results, we believe that our method constitutes a significant step toward the automation of the lightcurve classification pipeline.

Key words: methods: data analysis – stars: statistics

1. INTRODUCTION

Automatic classification of variable stars has received substantial attention in the research community in recent years (Debosscher et al. 2007; Kim et al. 2009, 2011, 2014; Wachman et al. 2009, pp. 489–505; Wang et al. 2010, p. 418; Bloom & Richards 2012; Richards et al. 2011; Bloom et al. 2012; Pichara et al. 2012a; Pichara & Protopapas 2013; Masci et al. 2014; Nun et al. 2014; Babu & Mahabal 2015; Hanif & Protopapas 2015; Neff et al. 2015). Achieving a good performance with these methods depends strongly on the way lightcurves are represented. Lightcurves are commonly represented as a vector of many statistical descriptors called features, which aim to measure a particular characteristic of the lightcurve. Feature calculation is intensive in computing resources, the development of new features requires a lot of research effort, and new features do not guarantee better classification performance. The upcoming and ongoing deep-sky surveys such as Pan-STARRS (Kaiser et al. 2002), LSST (Ivezic & the LSST Science Council 2011), and SkyMapper (Keller et al. 2007) are creating immense amounts of data, which makes the development of automatic and scalable analysis tools an important task for the astronomical community. Today, lightcurve representation is not entirely automatic: algorithms that extract lightcurve features are designed by astronomers and have to be manually tuned up for every new survey.

Most of the automatic classification tools coming from the machine learning community are very effective in the sense that they can produce accurate results and work very fast in the classification stage (after the training phase). However, results from classification algorithms are highly dependent on the way the data are represented, and a lot of effort is put into designing features to express lightcurves. For example, Kim et al. (2011) used a support vector machine (SVM) to classify variable stars, previously defining a set of time series descriptors to be used as features in the classification model. In a later work, Pichara et al. (2012a) made an important improvement in accuracy thanks to the inclusion of new features arising from a continuous autoregressive model. Huijse et al. (2012) made an improvement in classification of periodic stars by using an information theoretic approach to estimate periodicities. Nun et al. (2014) devised a method to detect anomalies in astronomical catalogs by using the results of a Random Forest classification as input for a Bayesian network.

The data representation problem arises in most fields that deal with data like time series and images since the complexity and size of the data usually make them unsuitable as direct input to any classification algorithm. To deal with this issue, the machine learning community propose a new way of representing data: unsupervised feature learning. This method aims to use unlabeled data to train a model that can then be used to transform data of the same kind to a new representation suitable for classification tasks. This process of transforming data from their raw form to another form is known as encoding. The development of unsupervised feature learning started with the objective of finding a good representation of images that could serve as input for learning algorithms. While the goal in most works is similar, approaches vary in nature. Olshausen & Field (1996) use sparse coding to represent an image, Bell & Sejnowski (1997) base their approach on signal analysis, Hinton & Salakhutdinov (2006) use models based on neural networks, while Coates & Ng (2012, p. 561) follow a clustering-based method.

We build on the ideas in Coates & Ng (2012, p. 561) even though their proposed method is not well established for time series. We make substantial modifications to their approach to

get meaningful results while using lightcurve data instead of images. These modifications have resulted in a new unsupervised learning method for lightcurves and time series in general. Our method is based on the clustering of hundreds of thousands of lightcurve subsequences, which allows us to find the most common and representative patterns in large amounts of data. The results of the clustering step are then used to transform lightcurves of a labeled set to a representation suitable for machine learning algorithms.

The purpose of this work is to introduce unsupervised feature learning as a strong alternative to expert-designed features that have traditionally been used for lightcurve representation in the context of automatic classification. The performance of classification models trained with data from our method is as good as and in some cases better than classifiers trained using the traditional lightcurve representation.

The remainder of this paper is organized as follows: Section 2 gives an account of the previous work in feature design for variable stars and the field of unsupervised feature learning, Section 3 introduces the relevant background theory for this work, Section 4 gives a detailed account of our methodology, and Section 5 presents the lightcurve catalogs and training sets used in this work. Section 6 discusses some implementation details and we show our results in Section 7. We give a brief run-time analysis in Section 8 and state the conclusions of our work in Section 9.

2. RELATED WORK

Automatic classification of lightcurves is currently performed by first transforming each lightcurve to a vector of many statistical descriptors, commonly called features, and then by training a learning algorithm. These features try to capture characteristics related to variability and periodicity, among others. Debosscher et al. (2007) represented a lightcurve as a vector of 28 parameters derived mainly from periodicity analysis. Kim et al. (2009) introduced the Anderson–Darling test in their method to de-trend lightcurves, which tests whether a given lightcurve can be said to be drawn from a normal distribution. This test has been included as a lightcurve feature in later work. Richards et al. (2011) introduced features that measure aspects like kurtosis, skewness, amplitude, deviation from the mean magnitude, linear slope, and many features extracted from periodicity analysis using the Lomb–Scargle periodogram. Kim et al. (2011) designed features to measure variability and dispersion and introduced the use of two photometric bands for some calculations. Pichara et al. (2012b) proposed the use of the continuous autoregressive model to strengthen the analysis of irregularly sampled lightcurves. Huijse et al. (2012) estimated periodicities with an algorithm based on information theory. Kim et al. (2014) introduced more features that relate variability and quartile analysis. Nun et al. (2015) designed a library that aims to facilitate feature extraction for astronomical lightcurves, which includes a compendium of features utilized throughout the recent literature. The design of all features for lightcurve representation that exist today has been the result of many years of research effort.

The tremendous amount of effort required to design new features has driven the focus of many research communities tackling other classification problems away from feature design and toward an unsupervised feature learning approach. Unsupervised feature learning models first emerged in the computer vision community as an effort to find a compact vector representation of images (Olshausen & Field 1996). Many of the models have since been adapted to work with time series data like speech, music, stock prices, and sensor readings. The results are varied, with some unsupervised learning approaches clearly improving the state-of-the-art performance on benchmark data sets. Sparse coding (Olshausen & Field 1996; Lee et al. 2006, p. 801), a methodology that aims to learn a set of overcomplete bases that can be used to represent data efficiently, was used by Grosse et al. (2007) for audio classification. Another common model that has been employed to solve time-series problems is the Restricted Boltzmann Machine (RBM, Hinton & Salakhutdinov 2006; Hinton et al. 2006; Larochelle & Bengio 2008). The RBM is a model that learns a distribution over its input data and is represented by an undirected bipartite graph. The weight matrix W, which describes the connections between nodes in the graph, can be used to transform data to a lower dimensional representation. This model has been used with success as a replacement for Gaussian mixtures in the discretization step required for hidden Markov models for audio classification (Dahl et al. 2012; Mohamed et al. 2012). Jaitly & Hinton (2011) used raw speech data as input for an RBM with success. Some variations of the RBM like the mean-covariance RBM (Krizhevsky et al. 2010; Ranzato & Hinton 2010) have also been used to improve on audio classification benchmarks (Dahl et al. 2010, p. 469).

Other, somewhat less popular unsupervised feature learning models that have been used with success in time series problems are the recurrent neural network (RNN, Hüsken & Stagge 2003), the Autoencoder (AE, Hinton & Salakhutdinov 2006; Poultney et al. 2006, p. 1137; Bengio 2009) and clustering approaches (Coates & Ng 2012, p. 561). The RNN is essentially a neural network in which the outputs are connected back to the inputs. It has been used with success in replacing both the Gaussian mixture and the hidden Markov model in the traditional audio classification pipeline (Graves et al. 2013). The AE is a neural network that tries to model the identity function of its input data. The weights in the network are adjusted during training to make the network's output as close as possible to its input. Längkvist & Loutfi (2012) use a modified version of the AE to perform unsupervised feature learning on sensor data, outperforming the best classification results obtained with expert-designed features. In clustering-based unsupervised feature learning, data are transformed into a new representation as a function of both the data and the most common data patterns found during clustering. Nam (2012) employs a clustering-based approach in combination with other models to perform music classification.

Due to the complexity of time series data, most of the works listed above still tackle unsupervised feature learning with the aid of some form of preprocessing, which requires both computational time and domain expertise. Raw time series data have been used with success in a limited number of problems, most notably by Jaitly & Hinton (2011). The previously mentioned models, on the other hand, are not designed to deal with the kind of time series that are common in astronomical surveys. Lightcurves are not sampled uniformly, so they have different numbers of observations for a fixed time frame. These characteristics of the data make them unsuitable as input for neural network-based models like the RBM and the AE, sparse coding, and most models that assume that the input is a vector

of a fixed size. Sensor data, digital sound, and stock prices do not have this problem, since they are sampled uniformly. Given the massive amounts of astronomical data to be collected in future surveys, the development of an automated pipeline for raw data analysis with minimal preprocessing is a priority.

3. BACKGROUND THEORY

Unsupervised feature learning algorithms, like the ones described in the previous section, work by learning a model from the usually vast amounts of unlabeled data available; this model can then be used to transform data to a representation suitable for machine learning tasks. The way we model the data in our feature learning approach is through a large set of representative local patterns that cover common occurrences in the lightcurves. To find these patterns, we run a clustering algorithm on a large set of unlabeled lightcurve subsequences, and then consider the representatives of each cluster found as a pattern to be included in our model.

When clustering any data, the measure used to evaluate the similarity between data points is of extreme importance to the quality of the results. In the domain of time series, the use of standard similarity measures like Euclidean distance and L_P norms, in general, is not suitable. Astronomical lightcurves are unevenly sampled, and thus the time series under comparison are rarely of the same length, so the Euclidean distance is not even well defined for the comparison of this kind of data. To solve this problem, "elastic measures" that tolerate uneven sampling and time series of different lengths have been proposed (Berndt & Clifford 1994; Chen et al. 2005). Serrà & Arcos (2014) have found the Time Warp Edit Distance (TWED, Marteau 2009) to be one of the most powerful and flexible for the case of unevenly sampled time series. Given that it allows for a meaningful comparison between any pair of time series of different lengths with even or uneven sampling, we use the TWED as the similarity measure for lightcurves in our experiments. The TWED is based on the Levenshtein distance (LD, Levenshtein 1966), commonly known as Edit Distance, which was initially defined as a measure to assess the similarity between two strings of characters and has been adapted to work with time series.

The use of the TWED as the similarity measure for lightcurve comparison poses an additional challenge for our lightcurve clustering; most clustering algorithms do not allow for the use of an arbitrary function to compare the input data. *K*-Means, for example, which has been used in previous unsupervised feature learning work, is designed to work with the Euclidean distance and no other measure. Modified versions of *K*-Means have been used to cluster lightcurves using measures like cross-correlation (Rebbapragada et al. 2009), but these modifications make the algorithm at least an order of magnitude slower. An additional disadvantage is that the number of clusters, K, has to be specified as input. Affinity Propagation (Frey & Dueck 2007) is a clustering algorithm that works with any input data as long as there is a similarity function defined for their comparison, which is exactly our case with the TWED. This algorithm has the additional advantage that it does not need an a priori specification of the number of clusters to find, and it defines a representative exemplar of each cluster. We use this set of exemplars as our lightcurve model.

What follows in this section is a detailed explanation of the Edit Distance for Time Series, followed by a definition of the TWED, and lastly a detailed description of the Affinity Propagation clustering algorithm.

3.1. Edit Distance for Time Series

The LD (Levenshtein 1966), commonly known as Edit Distance, is a distance metric used in many applications in computer science to assess the similarity between two strings of characters. The LD is defined as the smallest number of insertions, deletions, and substitutions required to change one string into another. The ideas behind LD have been extended for time series matching. What follows is a brief definition of the matching problem applied to time series.

Let U be the set of finite time series $U = \{X_1^p | p \in \mathbb{N}\}$, and X_1^p is a time series with discrete time index between 1 and p. Let x_i be the ith sample of time series X. We consider that $x_i \in S \times T$ where $S \subset \mathbb{R}$ embeds the time series values and $T \subset \mathbb{R}$ embeds the time variable. We say that $x_i = (m_{x_i}, t_{x_i})$ where $m_{x_i} \in S$ and $t_{x_i} \in T$, with $t_{x_i} > t_{x_j}$ whenever $i > j$ (time stamp strictly increases in the sequence of samples). X_i^j with $i < j$ is the sub-time series consisting of the ith through the jth sample (inclusive) of X. $|X|$ denotes the length (the number of samples) of X. Λ denotes the null sample.

An edit operation is a pair $(x, y) \neq (\Lambda, \Lambda)$ of time series samples, written $x \to y$. Time series Y results from the application of the edit operation $x \to y$ to time series X, written $X \Rightarrow Y$ via $x \to y$, if $X = \sigma x \tau$ and $Y = \sigma y \tau$ for some time series (both time series are the same except for subsets x and y). We call $x \to y$ a match operation if $x \neq \Lambda$ and $y \neq \Lambda$, a delete operation if $y = \Lambda$, and an insert operation if $x = \Lambda$. Similarly to the edit distance defined for strings, we can define $\delta(X, Y)$ as the similarity between any two time series X and Y of finite lengths p and q as

$$\delta(X_1^p, Y_1^q) = \min \begin{cases} \delta(X_1^{p-1}, Y_1^q) + \Gamma(x_p \to \Lambda) & \text{delete} \\ \delta(X_1^{p-1}, Y_1^{q-1}) + \Gamma(x_p \to y_q) & \text{match} \\ \delta(X_1^p, Y_1^{q-1}) + \Gamma(\Lambda \to y_q) & \text{insert} \end{cases}$$

where $p \geqslant 1$, $q \geqslant 1$, and Γ is an arbitrary cost function that assigns a non-negative real number $\Gamma(x \to y)$ to each edit operation $x \to y$.

It is worth pointing out that in the context of astronomical lightcurves, the notation X_1^p corresponds to a lightcurve with p observations, $x_i = (m_{x_i}, t_{x_i})$ is the ith observation with m_{xi} being its photometric magnitude and t_{xi} the observation time.

3.2. Time Warp Edit Distance

TWED is a similarity measure for time series based on the Edit Distance for time series but aims to provide an elastic metric for time series matching by taking the time differences into account when penalizing edit operations. TWED's edit operations are best understood as tools for superimposing two time series on a 2D graphical editor. Instead of *match*, *delete*, and *insert* operations, TWED defines the *match*, *delete-X*, and *delete-Y* operations:

1. *match*: the *match* operation (Figure 1(a)) consists of matching a segment (x_{i-1}, x_i) of X with a segment (y_{j-1}, y_j) of Y. In the graphical editor paradigm, the operation consists of clicking on the line that represents segment (x_{i-1}, x_i) and dragging and dropping it onto the line that represents segment (y_{j-1}, y_j). The cost of this

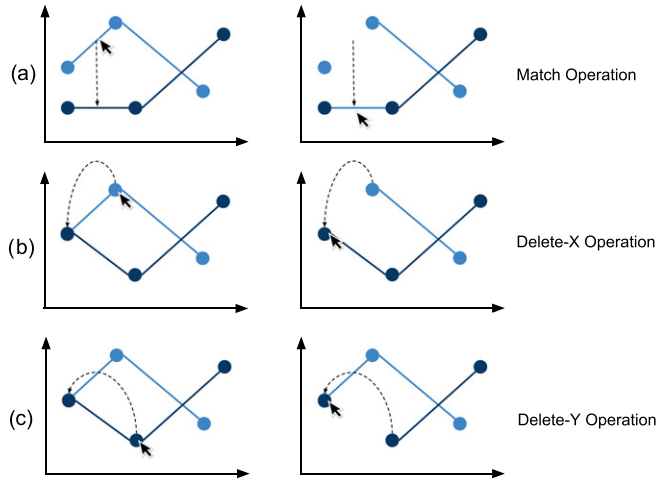

Figure 1. Edit operations in a graphical editor. Time series X and Y are depicted in light blue and dark blue, respectively.

operation is proportional to the sum of the distances between corresponding samples of the segments: $|y_j - x_i|$ and $|y_{j-1} - x_{i-1}|$.

2. *delete-X*: the *delete-X* operation (Figure 1(b)) consists of deleting a sample x_i. In the graphical editor paradigm, the operation consists of clicking on the point that represents sample x_i and dragging and dropping it onto the point that represents sample x_{i-1}. The cost associated with this delete operation is proportional to the length of the vector $(x_i - x_{i-1})$, to which a constant penalty λ is added.
3. *delete-Y*: just like the previous operation, the *delete-Y* operation (Figure 1(c)) consists of deleting a sample y_i. In the graphical editor paradigm, the operation consists of clicking on the point that represents sample y_i and dragging and dropping it onto the point that represents sample y_{i-1}. The cost associated with this delete operation is proportional to the length of the vector $(y_i - y_{i-1})$, to which a constant penalty λ is added.

The three edit operations are illustrated in Figure 1, following the idea of edit operations in a graphical editor paradigm.

The previous operations together with the definitions of Section 3.1 provide the basis for the definition of TWED:

$$\delta_{\lambda,\gamma}(X_1^p, Y_1^q) = \min\begin{cases}\delta_{\lambda,\gamma}(X_1^{p-1}, Y_1^q) + \Gamma_{\boldsymbol{x}} & \text{del-}X\\ \delta_{\lambda,\gamma}(X_1^{p-1}, Y_1^{q-1}) + \Gamma_{\boldsymbol{xy}} & \text{match}\\ \delta_{\lambda,\gamma}(X_1^p, Y_1^{q-1}) + \Gamma_{\boldsymbol{y}} & \text{del-}Y\end{cases}$$

where

$$\begin{aligned}\Gamma_{\boldsymbol{x}} &= |m_{x_p} - m_{x_{p-1}}| + \gamma|t_{x_p} - t_{x_{p-1}}| + \lambda\\ \Gamma_{\boldsymbol{xy}} &= |m_{x_p} - m_{y_q}| + \gamma|t_{x_p} - t_{y_q}|\\ &\quad + |m_{x_{p-1}} - m_{y_{q-1}}| + \gamma|t_{x_{p-1}} - t_{y_{q-1}}|\\ \Gamma_{\boldsymbol{y}} &= |m_{y_q} - m_{y_{q-1}}| + \gamma|t_{y_q} - t_{y_{q-1}}| + \lambda.\end{aligned}$$

It is important to note that parameter γ controls the "elasticity" of TWED: the higher it is, the higher the penalties related to time stamp differences. Like many proposed measures of time series similarity, TWED is calculated with a simple dynamic programming algorithm with running time $O(pq)$. The recursion is initialized to $\delta_{\lambda,\gamma}(X_1^i, Y_1^1) = \infty$, $\forall i > 1$; $\delta_{\lambda,\gamma}(X_1^1, Y_1^j) = \infty$, $\forall j > 1$, and $\delta_{\lambda,\gamma}(X_1^1, Y_1^1) = 1$.

To better understand TWED, consider the following example where two match operations are performed in total. Let X_1^3 and Y_1^3 be two time series with three samples each, $X = \{(1, 3), (3, 6), (8, 8)\}$ and $Y = \{(2, 1), (5, 8), (9, 7)\}$. Let $\lambda = \gamma = 1$. We have

$$\delta_{\lambda,\gamma}(X_1^3, Y_1^3) = \min\begin{cases}\delta_{\lambda,\gamma}(X_1^2, Y_1^3) + \Gamma_{\boldsymbol{x}}\\ \delta_{\lambda,\gamma}(X_1^2, Y_1^2) + \Gamma_{\boldsymbol{xy}}\\ \delta_{\lambda,\gamma}(X_1^3, Y_1^2) + \Gamma_{\boldsymbol{y}}\end{cases}$$

where

$$\begin{aligned}\Gamma_{\boldsymbol{x}} &= |m_{x_3} - m_{x_2}| + \gamma|t_{x_3} - t_{x_2}| + \lambda\\ &= |8 - 6| + 1 \times |8 - 3| + 1 = 8\\ \Gamma_{\boldsymbol{xy}} &= |m_{x_3} - m_{y_3}| + \gamma|t_{x_3} - t_{y_3}| + |m_{x_2} - m_{y_2}| + \gamma|t_{x_2} - t_{y_2}|\\ &= |8 - 7| + 1 \times |8 - 9| + |6 - 8| + 1 \times |3 - 5| = 6\\ \Gamma_{\boldsymbol{y}} &= |m_{y_3} - m_{y_2}| + \gamma|t_{y_3} - t_{y_2}| + \lambda\\ &= |7 - 8| + 1 \times |9 - 5| + 1 = 6\end{aligned}$$

so

$$\delta_{\lambda,\gamma}(X_1^3, Y_1^3) = \min\begin{cases}\delta_{\lambda,\gamma}(X_1^2, Y_1^3) + 8 & \text{del-}X\\ \delta_{\lambda,\gamma}(X_1^2, Y_1^2) + 6 & \text{match}\\ \delta_{\lambda,\gamma}(X_1^3, Y_1^2) + 6 & \text{del-}Y\end{cases}$$

We then calculate $\delta_{\lambda,\gamma}(X_1^2, Y_1^3)$, $\delta_{\lambda,\gamma}(X_1^3, Y_1^2)$ and $\delta_{\lambda,\gamma}(X_1^2, Y_1^2)$:

$$\begin{aligned}\delta_{\lambda,\gamma}(X_1^2, Y_1^2) &= \min\begin{cases}\delta_{\lambda,\gamma}(X_1^1, Y_1^2) + \Gamma_{\boldsymbol{x}} & \text{del-}X\\ \delta_{\lambda,\gamma}(X_1^1, Y_1^1) + \Gamma_{\boldsymbol{xy}} & \text{match}\\ \delta_{\lambda,\gamma}(X_1^2, Y_1^1) + \Gamma_{\boldsymbol{y}} & \text{del-}Y\end{cases}\\ &= \min\begin{cases}\infty\\ 0 + \Gamma_{\boldsymbol{xy}}\\ \infty\end{cases}\\ \Gamma_{\boldsymbol{xy}} &= |m_{x_2} - m_{y_2}| + \gamma|t_{x_2} - t_{y_2}| + |m_{x_1} - m_{y_1}|\\ &\quad + \gamma|t_{x_1} - t_{y_1}|\\ &= |6 - 8| + 1 \times |3 - 5| + |3 - 1| + 1 \times |1 - 2|\\ &= 7\end{aligned}$$

so

$$\begin{aligned}\delta_{\lambda,\gamma}(X_1^2, Y_1^3) &= \min\begin{cases}\delta_{\lambda,\gamma}(X_1^1, Y_1^3) + \Gamma_{\boldsymbol{x}} & \text{del-}X\\ \delta_{\lambda,\gamma}(X_1^1, Y_1^2) + \Gamma_{\boldsymbol{xy}} & \text{match}\\ \delta_{\lambda,\gamma}(X_1^2, Y_1^2) + \Gamma_{\boldsymbol{y}} & \text{del-}Y\end{cases}\\ &= \min\begin{cases}\infty\\ \infty\\ \delta_{\lambda,\gamma}(X_1^2, Y_1^2) + 6\end{cases}\\ &= 13\\ \delta_{\lambda,\gamma}(X_1^3, Y_1^2) &= \min\begin{cases}\delta_{\lambda,\gamma}(X_1^2, Y_1^2) + \Gamma_{\boldsymbol{x}} & \text{del-}X\\ \delta_{\lambda,\gamma}(X_1^2, Y_1^1) + \Gamma_{\boldsymbol{xy}} & \text{match}\\ \delta_{\lambda,\gamma}(X_1^3, Y_1^1) + \Gamma_{\boldsymbol{y}} & \text{del-}Y\end{cases}\\ &= \min\begin{cases}\delta_{\lambda,\gamma}(X_1^2, Y_1^2) + 8\\ \infty\\ \infty\end{cases}\\ &= 15\end{aligned}$$

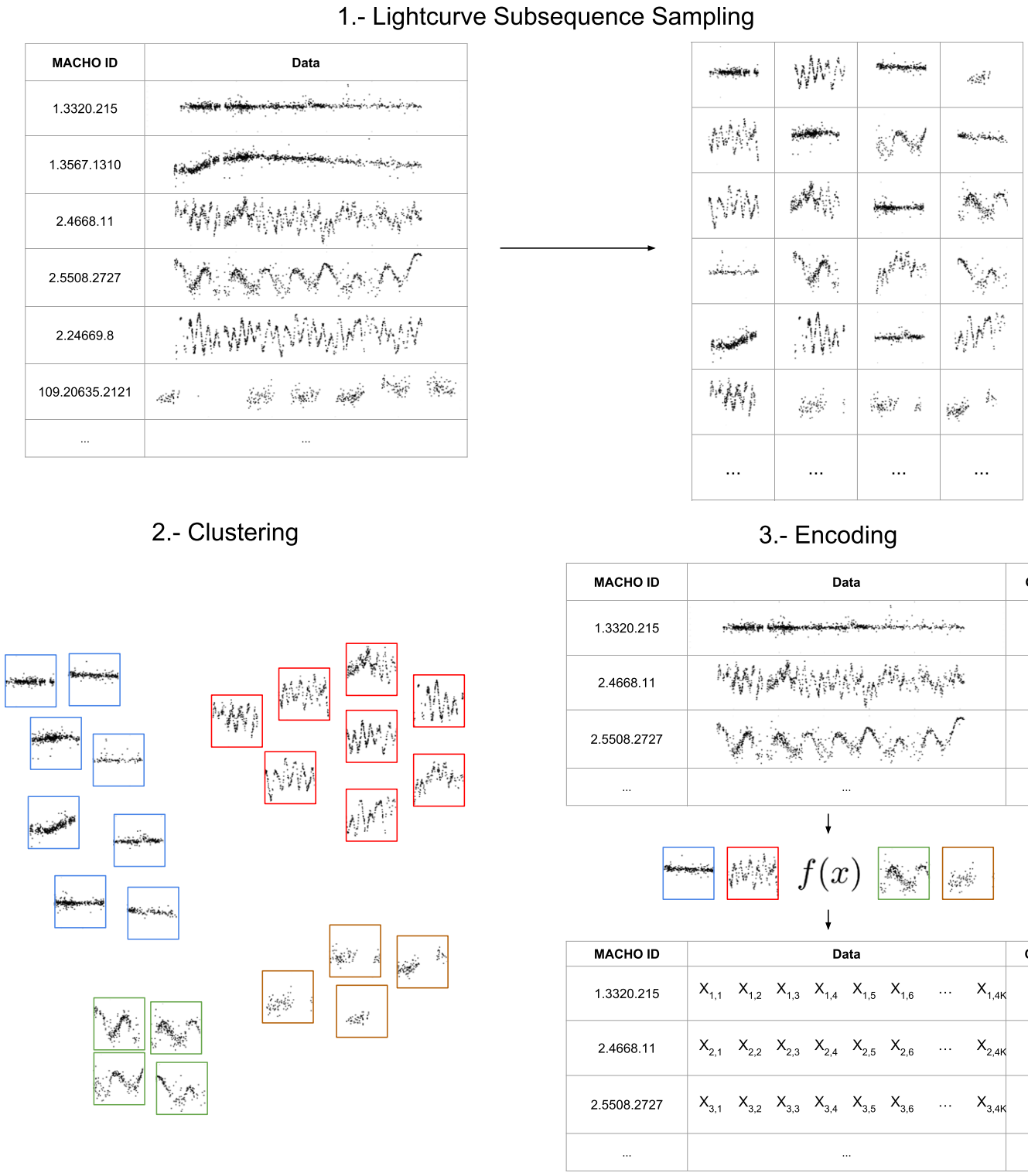

Figure 2. Illustration of an overview of the method: in the first step, we draw random subsequences from lightcurves to form a large set of lightcurve fragments. The second step consists of clustering these fragments with the Affinity Propagation algorithm. The third step consists of using the representative exemplars found during clustering to encode a training set of labeled lightcurves to a new representation more suitable for automatic classification tasks.

and finally

$$\delta_{\lambda,\gamma}(X_1^3, Y_1^3) = \min \begin{cases} 13 + 8 \\ 7 + 6 \\ 15 + 6 \end{cases} = 13.$$

The distance between X_1^3 and Y_1^3 is 13. If we were to calculate the TWED between two identical time series, the matching cost Γ_{xy} would be zero at each step. Is it easy to see then that the TWED between two identical time series is zero since at each step the match operation of zero cost would be chosen.

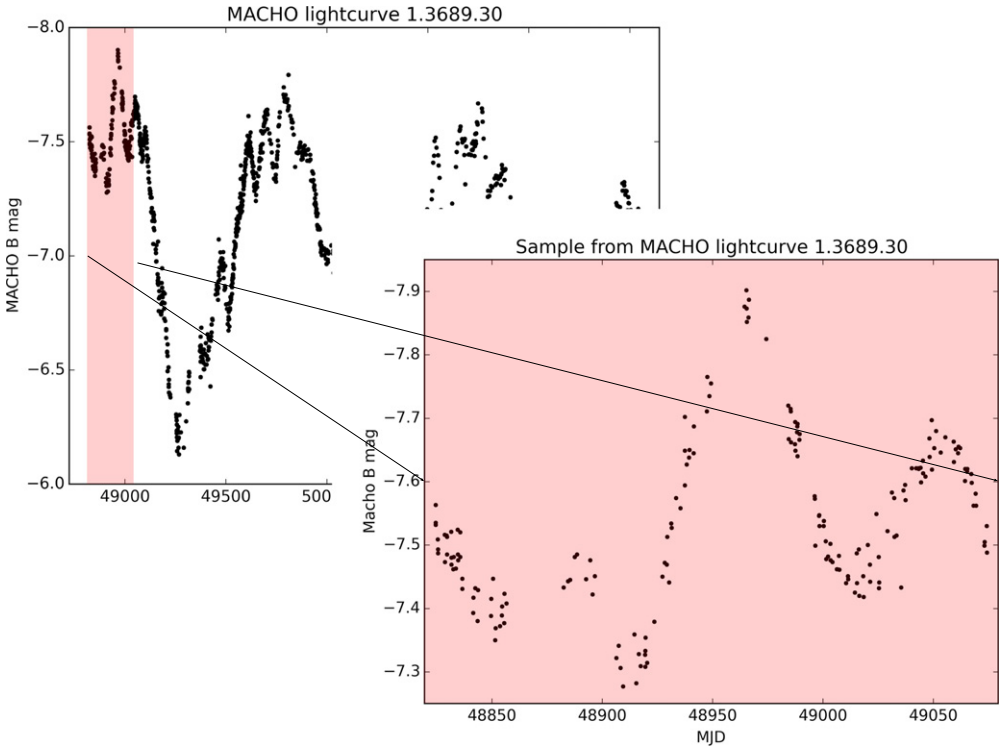

Figure 3. Lightcurve subsequence sampling. We sample a subsequence of the lightcurve by extracting all the observations in a given time window (translucent red), t_w.

3.3. Affinity Propagation

Affinity Propagation (Frey & Dueck 2007) is a clustering algorithm that aims to find representative exemplars from its input data. This algorithm views each data point as a node in a network, and recursively transmits real-valued messages along the edges of the network until a satisfactory set of exemplar points emerges. The magnitude of the transmitted messages reflects the "affinity" that one data point has for choosing another point as its exemplar.

The algorithm input is a matrix of real-valued similarities between data points, where $s(i, k)$ is the similarity between the data points with indices i and k. A higher value of $s(i, k)$ reflects a higher similarity. This measure is usually set to the negative Euclidean distance (distant points get low similarities), but the method can be applied to any arbitrary similarity measure. The values along the diagonal of the similarity matrix, $s(k, k)$, are called "preferences," and a larger value reflects a higher likelihood of being chosen as an exemplar during clustering. When no data point should be favored during clustering, like in our experiments, $s(k, k)$ should be set to a common value for all k. Another significant advantage of this algorithm besides the aforementioned flexibility is that in contrast to other common clustering algorithms like K-Means, Affinity Propagation does not require the number of clusters to be specified in advance. The number of clusters (number of exemplars) found is affected by both the values set for preferences and the message-passing procedure. In our experiments, we set the preferences to the median similarity between all points, which produces a moderate number of clusters (Frey & Dueck 2007). Another value used for the preferences is the minimum similarity, which produces a small number of clusters.

Data points exchange two different kinds of messages during clustering: "responsibility" $r(i, k)$ and "availability" $a(i, k)$. The first reflects the accumulated evidence for how good point k is to serve as an exemplar to point i, while the second reflects how appropriate it would be for point i to choose point k as its exemplar. The availabilities are initialized to zero: $a(i, k) = 0$. The responsibilities are then computed using the following update rule:

$$r(i, k) \leftarrow s(i, k) - \max_{k' s.t. k' \neq k} \{a(i, k') + s(i, k')\}.$$

This update rule should be seen as a competition between all candidate exemplars for ownership of a data point. The availability update rule, on the other hand, gathers evidence from data points as to whether a candidate exemplar would be a good exemplar:

$$a(i, k) \leftarrow \min\{0, r(k, k) + \sum_{i' s.t. i' \neq \{i,k\}} \max\{0, r(i', k)\}.$$

The availability $a(i, k)$ is set to the self-responsibility $r(k, k)$ plus the sum of the positive responsibilities that candidate exemplar k receives from other points. Only the positive portions of incoming responsibilities are added, because it is only necessary for a good exemplar to explain some data points well (positive responsibilities), regardless of how how poorly it explains other data points (negative responsibilities). The "self-availability" $a(k, k)$ is updated with the following rule:

$$a(k, k) \leftarrow \sum_{i' s.t. i' \neq k} \max\{0, r(i, k)\}.$$

This message reflects accumulated evidence that point k is an exemplar based on the positive responsibilities sent to candidate exemplar k from other points.

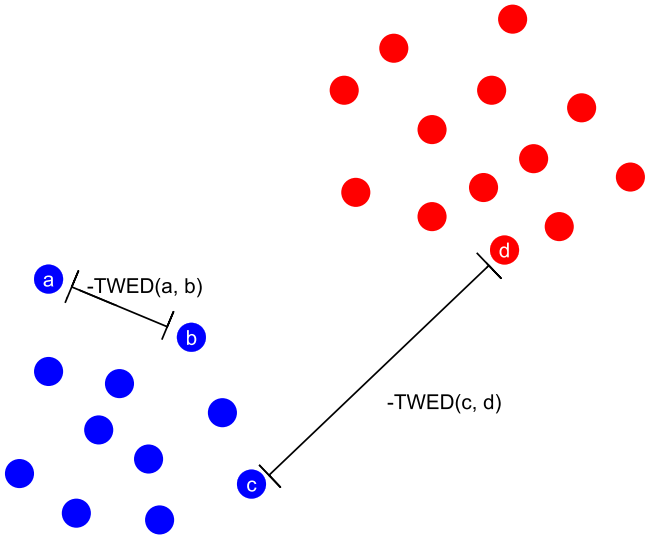

Figure 4. Lightcurve clustering. The lightcurve subsequences (represented by the colored dots) are grouped into clusters according to their affinity measure, which in this case is the negative TWED.

At any moment during Affinity Propagation, availabilities and responsibilities can be combined to identify exemplars. For point i, the value of k that maximizes $a(i, k) + r(i, k)$ either identifies point i as an exemplar if $k = i$, or identifies the data point that is the exemplar for point i. The message-passing procedure may be terminated after a fixed number of iterations, after changes in the messages fall below a threshold, or after the local decisions stay constant for some number of iterations.

4. DESCRIPTION OF THE METHOD

Our method draws from what was proposed in Coates & Ng (2012, p. 561) for the domain of images, with substantial modifications to make our new algorithm work well with lightcurves. As Keogh & Lin (2005) demonstrated, time-series subsequence clustering with K-Means and Euclidean distance will very seldom produce meaningful results. Furthermore, the Euclidean distance is not well defined for the comparison of two lightcurves since the two time series will rarely have the same length because they are not evenly sampled. To overcome this problem, we employ the TWED (Section 3.2) together with an appropriate clustering algorithm that works well with any similarity measure for its data, Affinity Propagation (Section 3.3).

Our algorithm consists of three main steps. In the first step, we randomly sample subsequences from lightcurves to form a large set of lightcurve fragments. The second step consists of clustering these fragments with the Affinity Propagation algorithm and the TWED similarity measure, both described in detail in Sections 3.3 and 3.2 respectively. The third step consists of using the representative exemplars, found during clustering, to encode a training set of labeled lightcurves to a new representation for the classification tasks. Figure 2 provides an illustrated overview of the process.

4.1. Lightcurve Subsequence Sampling

To get the data that we want to cluster, we randomly sample N subsequences of lightcurves from a given data set by extracting all the observations in a given time window, t_w. The idea behind sampling small time windows and not using the whole lightcurve is to force our model to capture local patterns in the data. This procedure is illustrated in Figure 3.

Table 1
Composition of the MACHO Training Set

	Class	Number of Objects
1	Non-nariable	3613
2	Quasar	17
3	Be Star	55
4	Cepheid	103
5	RR Lyrae	551
6	Eclipsing binary	42
7	Microlensing	173
8	Long-period variable	281

Table 2
Composition of the OGLE-III Training Set

	Class	Number of Objects
1	Cepheid	992
2	Type 2 Cepheid	476
3	RR Lyrae	971
4	Eclipsing binary	982
5	Delta Scuti	980
6	Long-period variable	957

4.2. Affinity Propagation Clustering

After collecting N lightcurve fragments from our data, we run the Affinity Propagation clustering algorithm with the set of fragments extracted in the first step as input data to find a set of representative lightcurve subsequences. This process is illustrated in Figure 4. The affinity measure used during clustering is the negative TWED: $-\delta_{\lambda,\gamma}(X_1^p, Y_1^q)$, where X_1^p and Y_1^q are two lightcurve fragments. We use the negative TWED since in that way a greater distance means a lesser degree of similarity. After the clustering is completed, we have a set of K representative exemplars from the data, which capture common local patterns occurring in the time series.

4.3. New Representation

With the K exemplars found during the clustering step, we use a feature mapping function f to map any lightcurve fragment to a new feature space. The idea is to encode any lightcurve fragment as a K-dimensional vector where each index of the vector will represent a degree of similarity between the lightcurve fragment and each of the K exemplars. Our choice of f is

$$f_k = \max\{0, \mu(\delta_{\lambda,\gamma}) - \delta_{\lambda,\gamma}(X_1^p, c^{(k)})\}$$

where X_1^p is a lightcurve fragment with p observations, $c^{(k)}$ is the kth exemplar, and $\mu(\delta_{\lambda,\gamma})$ is the average TWED between the fragment and all the other exemplars. This means that the value of any given index of the vector will be 0 if the distance to that exemplar is above average, and a positive value when the distance is below average. This value is larger when the

Table 3
Relevant Parameter Values

Name	Symbol	Value	Comments
Time window	t_w	250 days	We used 250 days to capture local patterns in the time series while allowing patterns from lightcurves with longer periodicities to also be captured (see Figure 6). We also considered using the autocorrelation function length (Kim et al. 2011) but the values of this feature for each class were too different for us to choose a good common value for all the data
Time step	t_s	10 days	Encoding will work best with as much overlap as possible between the adjacent lightcurve subsequences during the sliding window process (Section 4.3). Any redundant data will be eliminated through pooling, while no relevant patterns will be missed
Number of samples	N	20,000	The number of samples affects the performance of the clustering step significantly. We minimized the number of samples subject to still maintaining good classification performance. We consider 20,000 to be a sufficiently large number of samples while still keeping the computational time within reasonable bounds and allowing us to maintain our classification performance
TWED elasticity cost	γ	10^{-5}	We chose a relatively low penalty for this parameter to allow for higher "elasticity" when comparing lightcurve subsequences, in comparison to the values used in Marteau (2009)
TWED deletion cost	λ	0.5	We chose a mid-point penalty for this parameter so as not to bias the TWED toward matching operations when comparing lightcurve subsequences, in comparison to the values used in Marteau (2009)
Number of pooling regions	⋯	4	The number of regions over which to pool the data represents a trade-off between information preservation and dimensionality of the final representation. We chose four as the number of pools that would allow our representation to preserve the maximum amount of information while still maintaining a manageable dimensionality for the classification stage; this is the most common number used throughout the literature (Boureau et al. 2010; Coates & Ng 2012, p. 561). Empirically, we found four to work better in the classification stage

fragment is more similar to the exemplar. It is expected that roughly half the values in any given vector will be zero, which is a favorable condition for our classification procedure, detailed in Section 4.4.

Given this feature mapping function, we can now encode a complete lightcurve in our new representation by applying f to sequential fragments of the lightcurve. Specifically, given time step t_s and the time window t_w, the adjacent fragments are obtained by getting all the time series data in one window and then moving the time window by t_s, sliding the window across the whole lightcurve. It is worth noting that t_s is usually much smaller than t_w, so the extracted fragments overlap significantly. We extract adjacent fragments from each lightcurve until the sliding window reaches the end of the observations; this means that the number of fragments extracted is variable and depends on the length of the lightcurve. If M is the number of fragments extracted from a lightcurve, the final representation is of dimensions $\mathbb{R}^{M\times K}$. This process is illustrated in Figure 5.

This intermediate representation of a lightcurve is too large for use as direct input to any classification algorithm. To reduce the dimensionality of data while maintaining the maximum amount of information, it is a common practice to perform a procedure called feature pooling (Boureau et al. 2010). Pooling works by aggregating features extracted from a group of adjacent lightcurve fragments. Encoded fragments from windows that are adjacent or relatively close are also very similar, so finding a way to aggregate those features makes sense to reduce the dimensionality of data. In our experiments, we divide the final representation into four regions of equal size and aggregate the features inside each one. For each of the K features, we take the maximum value in each region, a procedure that is called max-pooling.

The final pooled representation of a complete lightcurve is a vector of size $4 \times K$, significantly smaller than the representation of size $M \times K$ that is obtained after the sliding window step. The number of regions over which to pool the data represents a trade-off between information preservation and dimensionality of the final representation. We chose four as the number of pools that would allow our representation to preserve the maximum amount of information while still maintaining a manageable dimensionality for the classification stage. Empirically, we found four to work better in the classification task.

4.4. Classification

The final training set is composed of all of the lightcurves encoded in our new representation together with their original labels. We use this data set to train a linear SVM classifier (Boser et al. 1992; Cortes & Vapnik 1995). The SVM is a classifier that tries to fit hyperplanes to data to separate classes. For an overview and discussion see Kim et al. (2012) and references therein.

5. DATA

The photometric data used in our experiments belong to two different catalogs, the Massive Compact Halo Object (MACHO) catalog and the Optical Gravitational Lensing Experiment (OGLE).

5.1. MACHO Catalog

The MACHO is a survey that observed the sky starting in 1992 July and ending in 1999 to detect microlensing events produced by Milky Way halo objects. Several tens of millions of stars were observed in the Large Magellanic Cloud (LMC), Small Magellanic Cloud (SMC), and Galactic bulge (Alcock et al. 1997).

5.2. OGLE-III Catalog of Variable Stars

The OGLE is a wide-field sky survey originally designed to search for microlensing events (Paczynski 1986). The brightness of more than 200 million stars in the Magellanic Clouds and the Galactic bulge is regularly monitored on a timescale of years. A by-product of these observations is an enormous

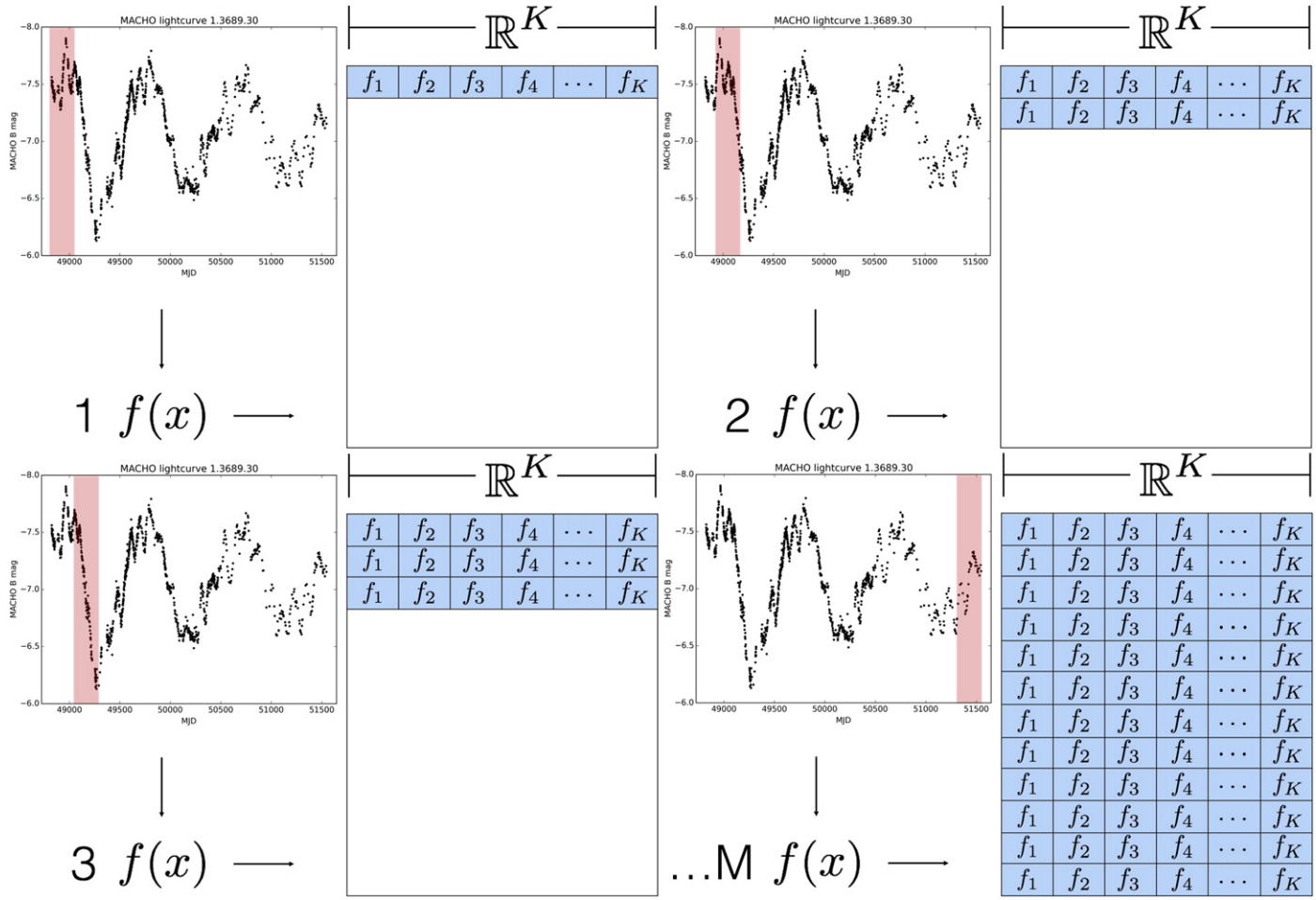

Figure 5. Sliding window process. The sliding window (translucent red) extracts a subsequence of the lightcurve at each step, which is encoded as a K-dimensional vector by our encoding function f. The window moves sequentially along the time axis, extracting and encoding one subsequence at each step.

database of photometric measurements. The OGLE-III Catalog of Variable Stars (Udalski et al. 2008) corresponds to the photometric data collected during the third phase of this survey, which began in 2001.

5.3. Training Sets

For our encoding and classification experiments we used subsets of both MACHO and OGLE surveys, corresponding to sets of labeled photometric data. The MACHO training set is composed of 4835 labeled observations (Kim et al. 2011); the OGLE training is composed of 5358 labeled variable objects from the OGLE-III Catalog of Variable Stars (Udalski et al. 2008). The per-class composition of both training sets is detailed in Tables 1 and 2. The OGLE training set was chosen as a subset of the most represented variable star classes in the catalog with the objective of creating a training set of comparable size to the MACHO data set.

6. IMPLEMENTATION

Our implementation uses minimal preprocessing: all lightcurves are adjusted to have zero mean and unit variance. To make our method robust to noise in the data, we discard the observations with high noise. More specifically, we remove all observations with errors bigger than three times the mean error of the lightcurve. Moreover, in the MACHO and OGLE data sets, the photometric errors are almost uniform across all measurements with the exception of a few outliers that we remove. This means that errors should not affect our result. For our lightcurve subsequence sampling step (Section 4.1) we sampled from thousands of unlabeled lightcurves. The parameters we used in our experiments are detailed in Table 3. The code for our experiments is available at https://github.com/cmackenziek/tsfl.

We used the Affinity Propagation and SVM implementations available in the machine learning library scikitlearn (Pedregosa et al. 2011). We also used the libraries numpy, scipy and pandas for data manipulation and efficient numerical computation (McKinney 2010; van der Walt et al. 2011).

7. EXPERIMENTAL RESULTS

In this section, we present the results obtained in our experiments. First, we present the results of the clustering step of our method, which we hope will help the reader gain qualitative insight into the inner workings of our algorithm. Then, we present the classification results on all the training sets described in Section 5 using two different classifiers and two methods of lightcurve representation: the classical expert-designed time series

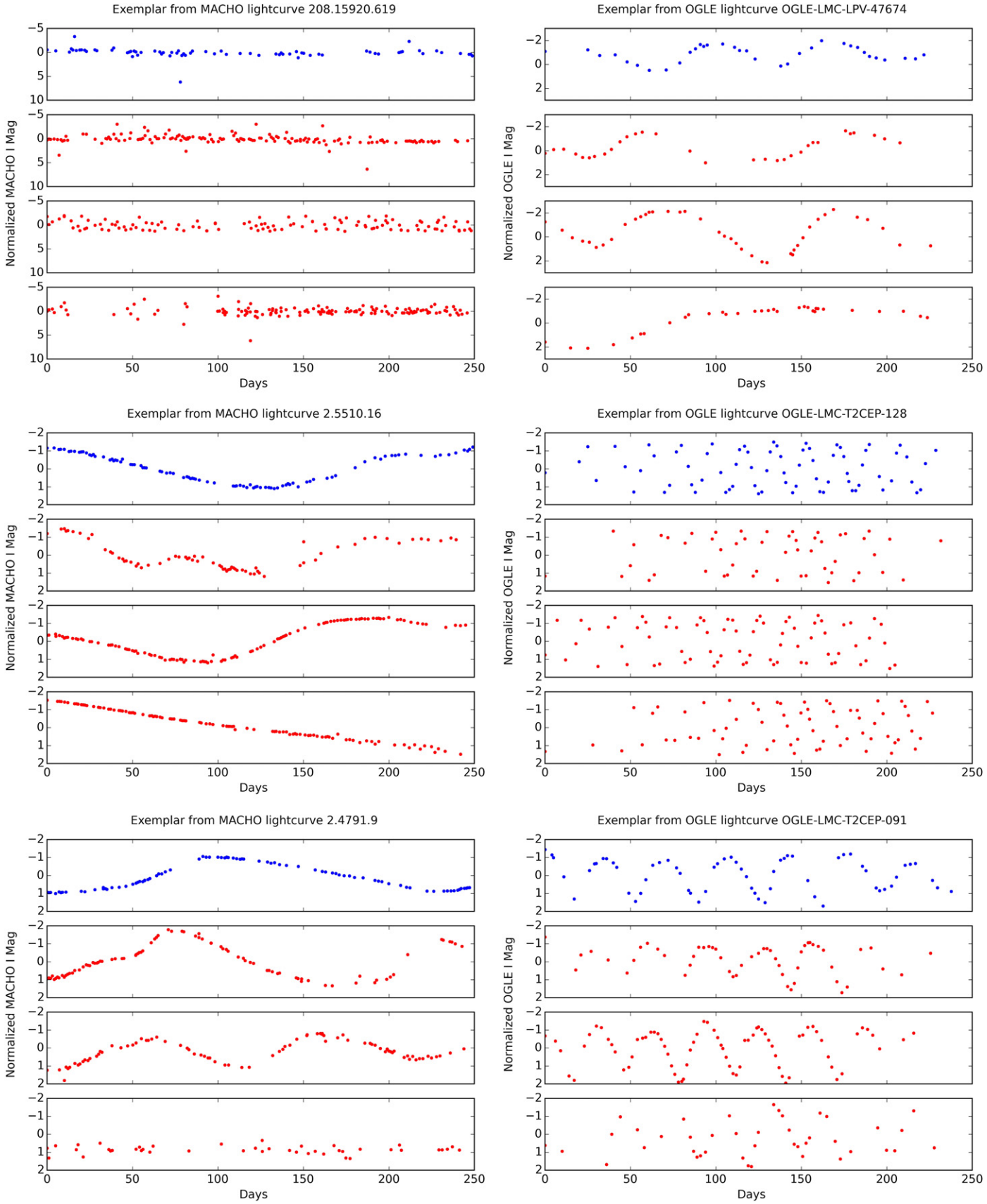

Figure 6. Cluster exemplars with members. Exemplars are lightcurve subsequences chosen by the clustering algorithm as the best representatives of their clusters. Each of the six plots shows an exemplar (plotted in blue) together with three other cluster members (plotted in red). We can appreciate how the clustering algorithm successfully groups similar lightcurve subsequences together.

Table 4
Classification *F*-Score on the MACHO Training Set

	Class	SVM Trained with LF	RF Trained with TSF	SVM Trained with TSF
1	Non-variable	0.991	0.991	0.875
2	Quasar	0.296	0.533	0.217
3	Be star	0.717	0.788	0.625
4	Cepheid	0.871	0.917	0.936
5	RR Lyrae	0.953	0.969	0.797
6	Eclipsing binary	0.780	0.763	0.725
7	Microlensing	0.980	0.974	0.468
8	Long-period variable	0.975	0.947	0.802
	Weighted average	0.975	0.978	0.807

Table 5
Classification *F*-Score on the OGLE-III Training Set

	Class	SVM Trained with LF	RF Trained with TSF	SVM Trained with TSF
1	Cepheid	0.835	0.737	0.555
2	Type 2 Cepheid	0.651	0.567	0.467
3	RR Lyrae	0.749	0.868	0.649
4	Eclipsing binary	0.862	0.602	0.458
5	Delta Scuti	0.817	0.656	0.656
6	Long-period variable	0.821	0.648	0.407
	Weighted average	0.821	0.696	0.696

Table 6
Number of Candidates per Class on MACHO Field 77

Class	Number of Candidates
Non-variable	382,306
Quasar	176
Be star	975
Cepheid	1,459
RR Lyrae	13,544
Eclipsing binary	85,099
Microlensing	26,231
Long-period variable	1,486

features and our learned features. Finally, we present an analysis of the classification relevance in terms of both types of features.

7.1. Clustering Results

Given that clustering aims to find groups of similar data, one would expect that clustering lightcurve subsequences would group similar patterns in the photometric data. Our results show that this is indeed the case. To show the results of the lightcurve subsequence clustering step described in Section 4.2, we provide plots of some of the learned exemplars together with some other lightcurve subsequences that are members of the same clusters. We can see some of the results in Figure 6: cluster exemplars are plotted in red together with some members of their respective clusters plotted in blue. We can see that the algorithm captures groups of similar subsequences

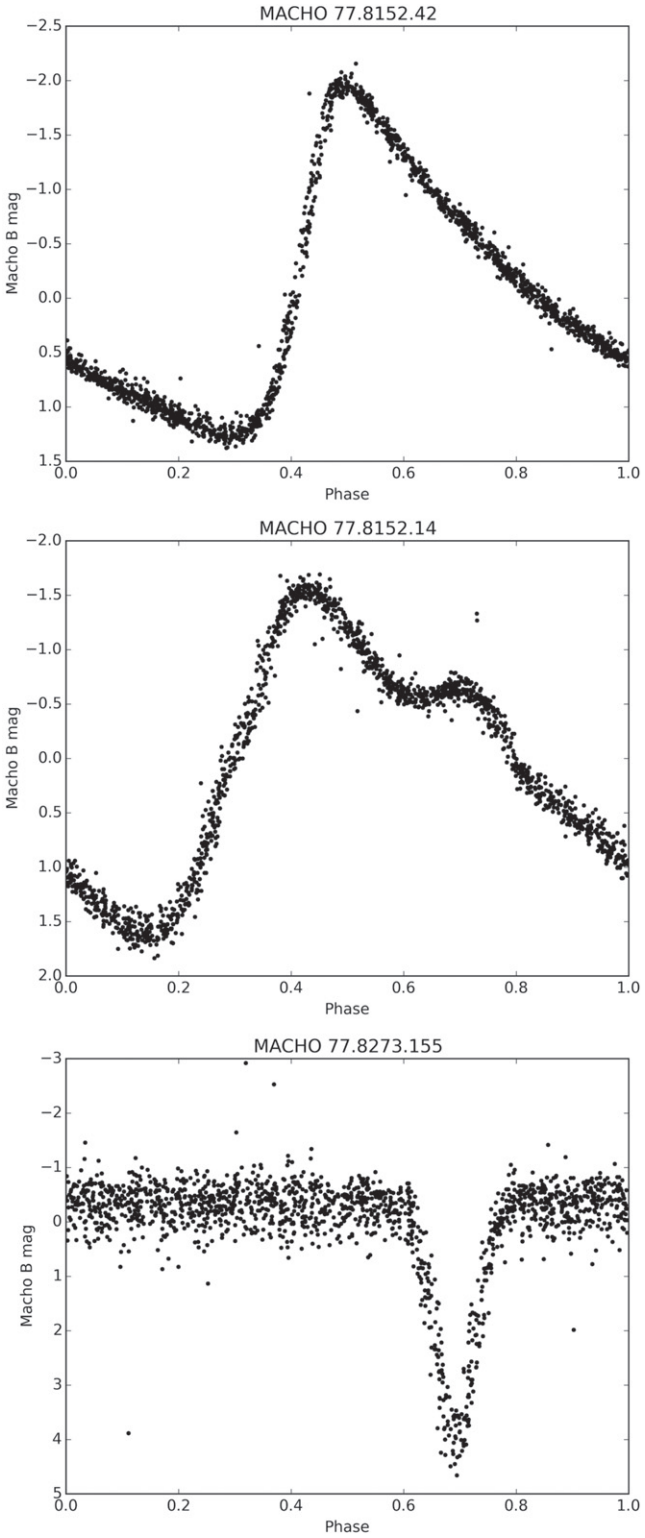

Figure 7. Examples of new variable star candidates. The lightcurves in the plots have been folded since they correspond to periodic stars. The first two lightcurves were classified as Cepheid while the third was classified as an eclipsing binary.

together. Lightcurve subsequences are grouped by important traits like variability and periodicity. This information is usually estimated with traditional features; our model can automatically group the lightcurve fragments without previously defining what the important criteria are. This fact

explains how the final encoding step will output relevant data that allow classifiers to distinguish correctly between light-curves of different classes: subsequences that exhibit different local photometric patterns will be encoded differently since they will be similar to a different subset of exemplars.

7.2. Classification Results from the Training Set

To evaluate the classification performance of a classifier trained on our learned features, we must obtain a benchmark with which to compare it. The logical benchmark for this task is the classification performance of a classifier using traditional time series features as input on the same training sets. Classifier performance is measured with a 10-fold stratified cross-validation *F*-score on each of the lightcurve classes of a given training set (or test set). Since the data produced by our feature learning method are high-dimensional and relatively sparse (each vector will have many zeroes by design, because of our encoding function *f*), we use a SVM (Cortes & Vapnik 1995) with a linear kernel as the classifier. To build the training sets for time series features, we applied the FATS Library (Nun et al. 2015), which has an exhaustive collection of time series features used throughout the literature. Traditionally, the classifier of choice for lightcurve data sets with time series features has been the Random Forest classifier (Breiman 2001); hence, we decided to compare our SVM with learned features against a Random Forest (RF) with the time series features. We also compared our SVM trained on learned features against an SVM trained on time series features. Tables 4 and 5 show the results for each training set. The acronym TSF refers to time series features, which are expert-designed features available in the FATS Library, and LF refers to learned features, which are the features we learn with our method. The SVM classifier performs as well as the Random Forest on the MACHO training set on many classes. Quasars are the only class where the SVM does not achieve comparable performance. We believe this to be due to the relatively low frequency of quasars in the whole training set, which is known to affect SVM classification performance. On the OGLE-III training set the SVM trained on learned features achieved superior results, only performing worse in one class. The first column details the variability class; the second column shows the result of an SVM classifier on 10-fold cross-validation with a linear SVM of both training sets. The learned features achieve a better overall classification performance than the time series features. All weighted averages are calculated using the relative frequency of each class of variable stars in the whole training set.

7.3. Classification Results from MACHO Field 77

In order to discover new variable star candidates, we classified 511,276 lightcurves from field 77 of the MACHO catalog. We found 128,970 variable star candidates; the per-class classification details are shown in Table 6. We cross-matched our variable star candidates with the SIMBAD Astronomical Database (Wenger et al. 2000) to filter out known candidates and found that 15,907 were already known and thus 113,873 are new. Figure 7 shows examples of our new candidates: the first two lightcurves were classified as Cepheid while the third was classified as an eclipsing binary. Our table of candidates is available for download at https://www.dropbox.com/s/fpsktd8aflelp7q/field77results_filtered.csv?dl=0. We will upload the catalog of our candidates to SIMBAD.

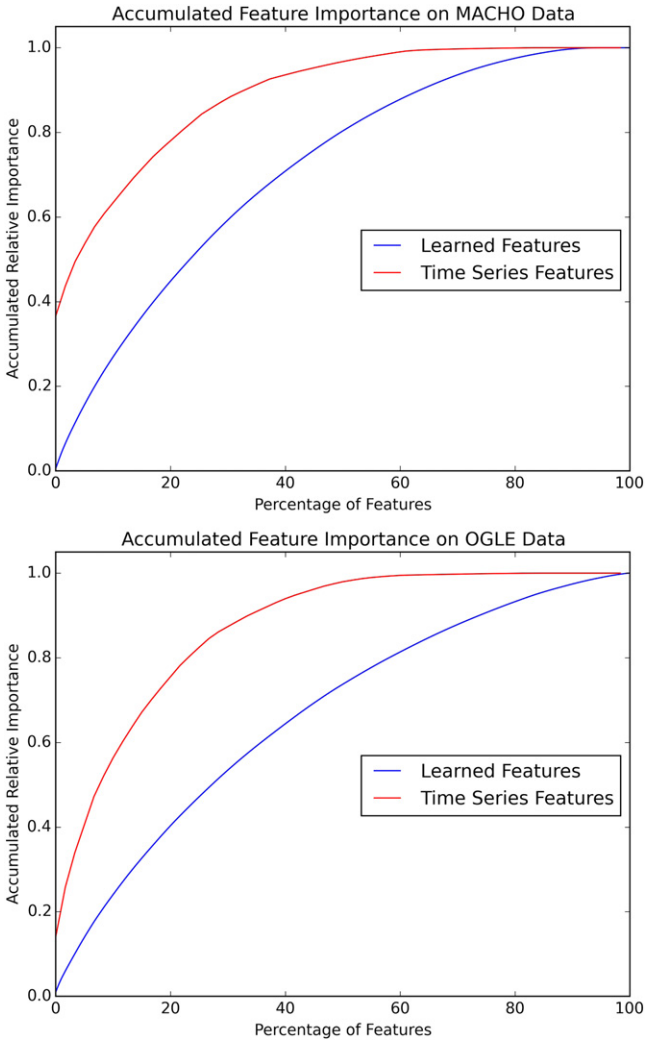

Figure 8. Cumulative sum of relative importance. A feature set that has lots of features that do not contribute much to classification (i.e., rarely get used by the SVM to separate between classes) would result in a plot where the cumulative sum reaches 1 with a lower percentage of features than a feature set where most of the features are relevant to the classification task for at least some of the classes.

7.4. Feature Importance

A very important aspect of a successful classification model is representing the data with relevant features that help the model distinguish between the different labeled data. With this in mind, it would be interesting to analyze how each of the features in a data set contributes to the final classification task. We performed a feature importance analysis on our learned features and time series features, using the hyperplane coefficients of an SVM with a linear kernel. The general idea behind using the hyperplane coefficients of an SVM is that the importance of a feature in separating between classes is proportional to the magnitude of its corresponding coefficient. A feature that completely separates the classes will have a coefficient of -1 or 1, while a completely irrelevant feature will have a coefficient of 0. A more detailed explanation of the theory behind this analysis is given by Guyon et al. (2002), who used SVM coefficients for selection of gene subsets.

Since our classification problems are multi-class, we get one separating hyperplane for each class. We defined the relative feature importance, *i*, as the sum of the absolute values of each

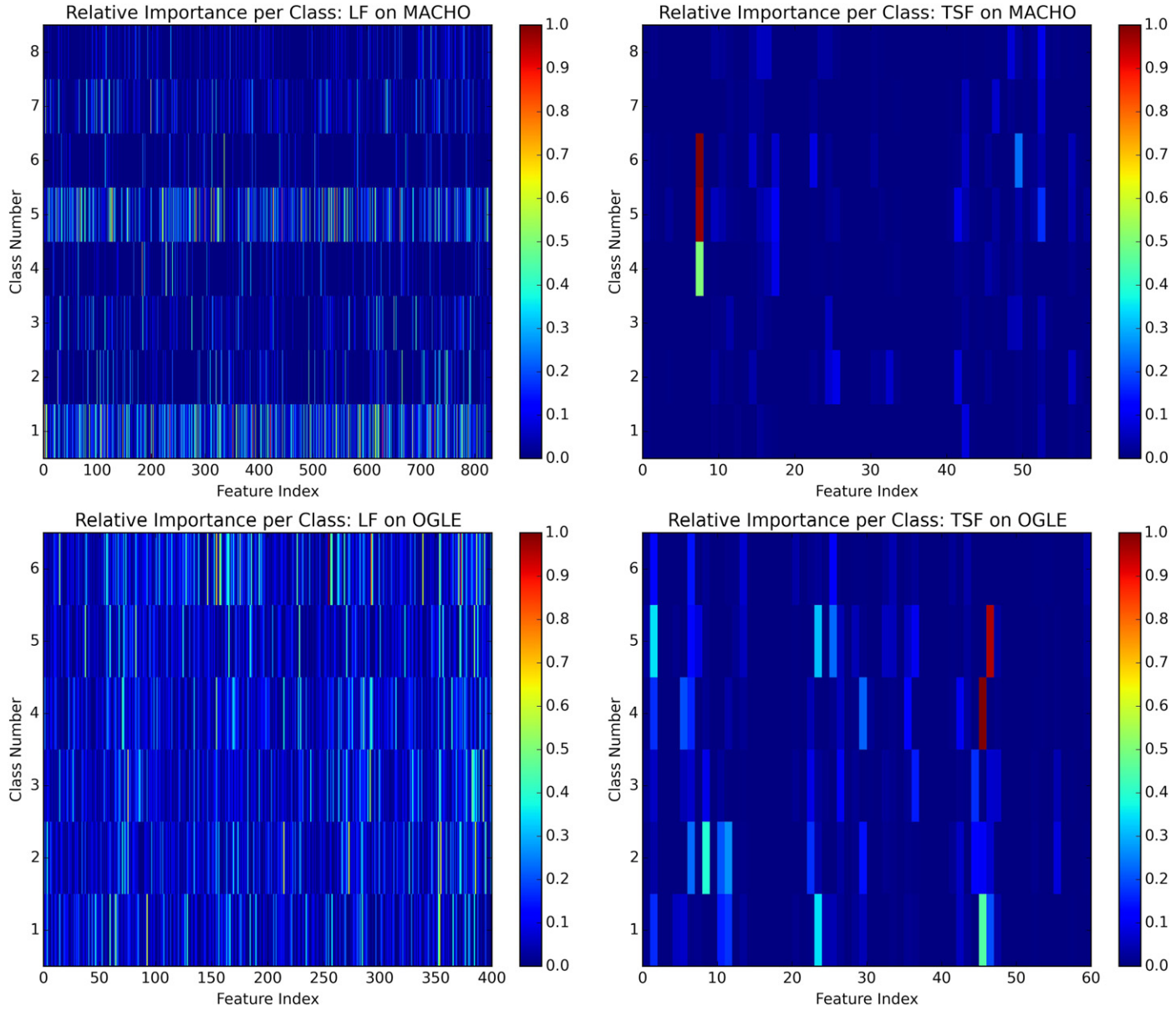

Figure 9. Relative importance per class. The two heatmaps show the relative importance of each feature of both training sets constructed with the MACHO and OGLE data. We can see in the figure that the contribution of each learned feature to classification is complex while the contribution of designed features is in many cases minimal, and classification with these features is largely based on the contribution of a few of the best features.

Table 7
Computational Run-time Details

Step	Run Time
Sampling (Section 4.1)	5 minutes
Clustering (Section 4.2)	10 minutes
Encoding (Section 4.3)	1.5–3 hr
Total	1.75–3.25 hr

Table 8
Average Encoding Time

Training Set	LF	TSF
MACHO	1.95 s	12.94 s
OGLE-III	0.86 s	7.70 s

of the feature's coefficients in each hyperplane, divided by the sum of the importance of all features:

$$i_k = \frac{\sum_{j=1}^{N} |w_k^j|}{\sum_{k=1}^{F} \sum_{j=1}^{N} |w_k^j|}$$

where w_k^j is the coefficient with index k of hyperplane j, N is the number of classes, and F the number of features. We can visualize at a high level how the two types of features contribute to classification by looking at Figure 8, which plots the cumulative sum of the relative feature importances versus the percentage of features being added. A feature set that has lots of features that do not contribute much to classification (i.e., rarely get used by the SVM to separate between classes) would result in a plot where the cumulative sum reaches 1 with

a low percentage of the features. On the other hand, a feature set where most of the features are relevant to the classification task for at least some of the classes would result in a plot where the cumulative sum reaches 1 with a high percentage of the features. This is the case for Figure 8, where it is evident that a great percentage of the time series features do not contribute much to classification.

Another way of analyzing the contribution of learned features to classification is by looking at the relative importance of each of the features in each of the hyperplanes that are learned in each class during the SVM training (a multi-class SVM learns one hyperplane to separate each class from all of the rest). We can see in Figure 9 that the contribution of learned features to classification is complex and that their contribution is different for each class: most of the features have very different relative importance values for each class. Time series features, on the other hand, rely heavily on a few features for classification: a clear example of this is the red color of one feature for classes 5 and 6 in the top right heatmap, while the other features look mostly dark blue (the lowest relative importance value).

8. COMPUTATIONAL RUN-TIME ANALYSIS

To address the scalability requirements of future astronomical surveys, it is important that analysis algorithms run within manageable time frames and scale well with an increase in the volume of input data. Table 7 shows the approximate run times for each step of our method described in Section 4. The two main algorithms on which we rely are Affinity Propagation and the TWED. The algorithmic complexity for Affinity Propagation is $O(N^2)$ where N is the number of points (lightcurve subsequences) being clustered, and the complexity of TWED is $O(pq)$ where p and q are the numbers of samples in each of the time series under comparison.

Our method is also significantly faster in transforming a lightcurve from its time series to its encoded vector representation. If we compare against calculating the time series features that we used for comparison in our experiments, we find that the speed gain is almost an order of magnitude, as shown in Table 8. One might argue that this is not a fair comparison since our method depends on the execution of previous steps, namely sampling and clustering, but that overhead is a constant cost that does not increase with the number of lightcurves to be transformed.

9. CONCLUSIONS

In this work, we have introduced a new way of modeling and representing lightcurve data as input for automatic classifiers. The method does not assume previous knowledge about the lightcurves or use any expert knowledge, unlike previous traditional methods that use a set of time series features specially designed by astronomers for the task of classification. The previous fact together with the possibility of leveraging the vast amount of information available in unlabeled data constitutes a big step toward a more automatic, flexible, and powerful classification pipeline. Our method works by extracting a large number of lightcurve subsequences from a given set of photometric data, which are then clustered to find common local patterns in the time series. Representatives of these common patterns, called exemplars, are then used to transform lightcurves of a labeled set into a new representation that can then be used to train an automatic classifier.

Our results show that this representation is as suitable for classification purposes as the traditional time series feature-based representation. Classifiers trained with our features perform as well as ones trained with expert-designed features, while the computational cost of our method is significantly lower. With our method we were able to find 113,873 new variable star candidates. Our hope is that the research community will hold feature learning methods as a valid alternative to lightcurve representation in future work since we have shown them to be a strong competitor to the expert-designed time series features. Our implementation code is readily available for others to download and build upon: users should try to adjust the parameters mentioned in Table 3 to suit their particular application.

While our method does not deal directly with errors in the photometric observations, the use of clustering makes our method robust regarding noise without specifically having to model errors. The intuition behind this is that the clustering stage finds common occurring patterns in the whole data set of fragments. Lightcurve subsequences significantly affected by random noise are not likely to be similar to many other subsequences, and thus will not be chosen as exemplars. The use of our method with highly noisy data would probably require the utilization of a similarity measure, which considers errors in its measurements. This could be done by comparing the distributions of the two subsequences, such as by χ^2 measure, the mutual information criteria, or simply by adding a penalty term that depends on the errors. The first two methods are computationally very expensive and not appropriate for the type of applications we are considering. The last is an ad-hoc measure and not statistically motivated, and it requires an extra optimization procedure (extra coefficient in TWED's operations) that will also impact the computational cost. Significant further research would be needed to develop a fast approximated optimization process, which is outside the scope of this work.

This work is supported by Vicerrectoría de Investigación (VRI) from Pontificia Universidad Católica de Chile, Institute of Applied Computer Science at Harvard University, and the Chilean Ministry for the Economy, Development, and Tourism's Programa Iniciativa Científica Milenio through grant P07-021-F, awarded to The Milky Way Millennium Nucleus. This research has made use of the SIMBAD database, operated at CDS, Strasbourg, France.

REFERENCES

Alcock, C., Allsman, R., Alves, D., et al. 1997, ApJ, 479, 119
Babu, G. J., & Mahabal, A. 2015, Skysurveys, Light Curves and Statistical Challenges, International Statistical Review, doi:10.1111/insr.12118
Bell, A. J., & Sejnowski, T. J. 1997, Vision Research, 37, 3327
Bengio, Y. 2009, Foundations and Trends in Machine Learning, 2, 1
Berndt, D. J., & Clifford, J. 1994, in the Twelfth Conf. on Artificial Intelligence (AAAI-94) KDD Workshop 10 (Seattle, WA), 359
Bloom, J., & Richards, J. 2012, in Advances in Machine Learning and Data Mining for Astronomy, ed. M. J. Way et al. (London: CRC), 89
Bloom, J., Richards, J., Nugent, P., et al. 2012, PASP, 124, 1175
Boser, B. E., Guyon, I. M., & Vapnik, V. N. 1992, in Proc. Fifth Annual Workshop on Computational Learning Theory (New York: ACM), 144
Boureau, Y.-L., Ponce, J., & LeCun, Y. 2010, in Proc. 27th Int. Conf. on Machine Learning (ICML-10), 111
Breiman, L. 2001, Machine Learning, 5

Chen, L., Özsu, M. T., & Oria, V. 2005, in Proc. 2005 ACM SIGMOD Int. Conf. on Management of Data (New York: ACM), 491
Coates, A., & Ng, A. Y. 2012, Neural Networks: Tricks of the Trade (Berlin: Springer)
Cortes, C., & Vapnik, V. 1995, Machine Learning, 20, 273
Dahl, G., Mohamed, A.-r., Hinton, G. E., et al. 2010, Advances in Neural Information Processing Systems (New York: Curran Associates, Inc.)
Dahl, G. E., Yu, D., Deng, L., & Acero, A. 2012, IEEE Transactions on Audio, Speech, and Language Processing, 20, 30
Debosscher, J., Sarro, L., Aerts, C., et al. 2007, A&A, 475, 1159
Frey, B. J., & Dueck, D. 2007, Sci, 315, 972
Graves, A., Mohamed, A.-r., & Hinton, G. 2013, in 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (Piscataway, NJ: IEEE), 6645
Grosse, R., Raina, R., Kwong, H., & Ng, A. 2007, in Proc. Twenty-Third Annual Conf. on Uncertainty in Artificial Intelligence (UAI-07) (Corvallis, OR: AUAI Press), 149
Guyon, I., Weston, J., Barnhill, S., & Vapnik, V. 2002, Machine Learning, 46, 389
Hanif, A., & Protopapas, P. 2015, MNRAS, 448, 390
Hinton, G. E., Osindero, S., & Teh, Y.-W. 2006, Neural Computation, 18, 1527
Hinton, G. E., & Salakhutdinov, R. R. 2006, Sci, 313, 504
Huijse, P., Estévez, P., Protopapas, P., et al. 2012, ITSP, 60, 5135
Hüsken, M., & Stagge, P. 2003, Neurocomputing, 50, 223
Ivezic, Z., & the LSST Science Council 2011, The LSST System Science Requirements Document, v5.1.3
Jaitly, N., & Hinton, G. 2011, in 2011 IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP) (Piscataway, NJ: IEEE), 5884
Kaiser, N., Aussel, H., Burke, B. E., et al. 2002, Proc. SPIE, 4836, 154
Keller, S. C., Schmidt, B. P., Bessell, M. S., et al. 2007, PASA, 24, 1
Keogh, E., & Lin, J. 2005, Knowledge and Information Systems, 8, 154
Kim, D.-W., Protopapas, P., Alcock, C., Byun, Y.-I., & Bianco, F. B. 2009, MNRAS, 397, 558
Kim, D.-W., Protopapas, P., Bailer-Jones, C. A., et al. 2014, A&A, 566, A43
Kim, D.-W., Protopapas, P., Byun, Y.-I., et al. 2011, ApJ, 735, 68
Kim, D.-W., Protopapas, P., Trichas, M., et al. 2012, ApJ, 747, 107
Krizhevsky, A., Hinton, G. E., et al. 2010, in Int. Conf. on Artificial Intelligence and Statistics, 621
Längkvist, M., & Loutfi, A. 2012, in NIPS Workshop on Deep Learning and Unsupervised Feature Learning
Larochelle, H., & Bengio, Y. 2008, in Proc. 25th Int. Conf. on Machine Learning (New York: ACM), 536
Lee, H., Battle, A., Raina, R., & Ng, A. Y. 2006, Advances in Neural Information Processing Systems (Cambridge, MA: MIT Press)
Levenshtein, V. 1966, SPhD, 10, 707
Marteau, P.-F. 2009, ITPAM, 31, 306
Masci, F. J., Hoffman, D. I., Grillmair, C. J., & Cutri, R. M. 2014, AJ, 148, 21
McKinney, W. 2010, in Proc. of the 9th Python in Science Conf. (SciPy), ed. S. vander Walt & J. Millman, 51
Mohamed, A.-r., Dahl, G. E., & Hinton, G. 2012, IEEE Transactions on Audio, Speech, and Language Processing, 20, 14
Nam, J. 2012, PhD thesis, Stanford Univ.
Neff, J., Wells, M., Geltz, S., & Brown, A. 2015, in Cambridge Workshop on Cool Stars, Stellar Systems, and the Sun 18, ed. G. van Belle & H. C. Harris, 879
Nun, I., Pichara, K., Protopapas, P., & Kim, D.-W. 2014, ApJ, 793, 23
Nun, I., Protopapas, P., Sim, B., et al. 2015, arXiv:1506.00010
Olshausen, B. A., & Field, D. J. 1996, Natur, 381, 607
Paczynski, B. 1986, ApJ, 304, 1
Pedregosa, F., Varoquaux, G., Gramfort, A., et al. 2011, Journal of Machine Learning Research, 12, 2825
Pichara, K., & Protopapas, P. 2013, ApJ, 777, 83
Pichara, K., Protopapas, P., Kim, D., Marquette, J., & Tisserand, P. 2012a, MNRAS, 18, 1
Pichara, K., Protopapas, P., Kim, D.-W., Marquette, J.-B., & Tisserand, P. 2012b, MNRAS, 427, 1284
Poultney, C., Chopra, S., Cun, Y. L., et al. 2006, Advances in Neural Information Processing Systems (Cambridge, MA: MIT Press)
Ranzato, M., & Hinton, G. E. 2010, in 2010 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (Piscataway, NJ: IEEE), 2551
Rebbapragada, U., Protopapas, P., Brodley, C. E., & Alcock, C. 2009, Machine Learning, 74, 281
Richards, J. W., Starr, D. L., Butler, N. R., et al. 2011, ApJ, 733, 10
Serrà, J., & Arcos, J. L. 2014, Knowledge-Based Systems, 67, 305
Udalski, A., Szymanski, M. K., Soszynski, I., & Poleski, R. 2008, AcA, 58, 69
van der Walt, S., Colbert, S. C., & Varoquaux, G. 2011, CoRR, arXiv:1102.1523
Wachman, G., Khardon, R., Protopapas, P., & Alcock, C. 2009, in Machine Learning and Knowledge Discovery in Databases, Vol. 5782, ed. W. Buntine et al. (Berlin: Springer)
Wang, Y., Khardon, R., & Protopapas, P. 2010, Machine Learning and Knowledge Discovery in Databases, Vol. 6323 (Berlin: Springer)
Wenger, M., Ochsenbein, F., Egret, D., et al. 2000, A&AS, 143, 9