PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE

SCHOOL OF ENGINEERING

CLUSTERING BASED FEATURE LEARNING ON VARIABLE STARS

CRISTÓBAL MACKENZIE KIESSLER

Thesis submitted to the Office of Research and Graduate Studies

in partial fulfillment of the requirements for the degree of

Master of Science in Engineering

Advisor:

KARIM PICHARA BAKSAI

Santiago de Chile, April 2016

PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE

SCHOOL OF ENGINEERING

CLUSTERING BASED FEATURE LEARNING ON VARIABLE STARS

CRISTÓBAL MACKENZIE KIESSLER

Members of the Committee:

KARIM PICHARA BAKSAI

CRISTIÁN TEJOS

JULIO PERTUZÉ SALAS

FRANZ BAUER

PAVLOS PROTOPAPAS

JAIME NAVÓN COHEN

Thesis submitted to the Office of Research and Graduate Studies

in partial fulfillment of the requirements for the degree of

Master of Science in Engineering

Santiago de Chile, April 2016

To my family, friends and Josefina

ACKNOWLEDGEMENTS

I would like to thank my advisor Karim Pichara and professor Pavlos Protopapas for all their support through my studies and research, for always having their doors open for brainstorming and for pushing me to do my best when my motivation was running low.

I would also like to thank my parents for always encouraging my curiosity, for giving me the opportunity to pursue my studies and for being supportive of every initiative, interest and opinion over the years.

I would also like to thank my lab partners Nicolás Castro and Andrés Riveros for keeping the work atmosphere relaxed but productive and always sharing their findings and ideas.

My final acknowledgements go out to the administrative and cleaning staff at the Computer Science Department, who work very hard every day to keep everything running smoothly. Their contribution to our work goes far beyond their job description. ¡Muchas gracias!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The success automatic of classification of variable stars depends 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 MACHO and OGLE datasets; the results show that our classification performance is as good 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 towards the automatization of the lightcurve classification pipeline.

Keywords: Astronomy, Variable Stars, Machine Learning, Data Mining

RESUMEN

El éxito de la clasificación automática de estrellas variables depende en gran medida de la representación de la curva de luz. Comúnmente, una curva de luz es representada como un vector de descriptores estadísticos diseñados por astrónomos llamados características. Estas características son costosas de calcular, requieren mucho tiempo de investigación para desarrollar y no garantizan un buen rendimiento de clasificación. Hoy en día la representación de curvas de luz no es automática; los algoritmos deben ser diseñados y ajustados para cada set de datos. La cantidad de datos astronómicos que se generará en el futuro requerirá de procesos de análisis automáticos y escalables. En este trabajo presentamos un algoritmo de aprendizaje de características diseñado para objetos variables. Nuestro método funciona a través de la extracción de un gran número de subsecuencias de curvas de luz, de las cuales se extraen subsecuencias representantes de los patrones más comunes a través de un algoritmo de clustering. Estos representantes son usados para transformar curvas de luz de un conjunto etiquetado a una representación que puede ser usada con un clasificador. El algoritmo propuesto aprende características de datos etiquetados y no etiquetados, lo que elimina el sesgo de usar solo datos etiquetados. Evaluamos nuestro método en las bases de datos MACHO y OGLE; los resultados muestran que nuestro rendimiento de clasificación es tan bueno como y en algunos casos mejor que el rendimiento que se logra usando las características tradicionales, mientras que el costo computacional es significativamente

menor. Con estos resultados prometedores, creemos que nuestro método constituye un paso significativo hacia la automatización de los procesos de clasificación de curvas de luz.

Palabras Claves: Astronomía, Estrellas Variables, Aprendizaje de Máquina, Minería de Datos

1. INTRODUCTION

1.1. Data Analysis in Astronomy

In order to study and understand the objects that make up our universe, astronomers must analyze ever-increasing amounts of data. Astronomical surveys, which are general images of a region of the sky which lack a particular observation target, are producing amounts of data that are challenging the existing algorithms astronomers use to analyze data in terms of scalability. A as a matter of example, the MACHO survey (Alcock et al., 1997), which observed three regions of the sky from 1992 to 1999, produced approximately 10 Terabytes of data. More recently, LSST (Matter, 2007) which is an ongoing survey expected to conclude in 2021, is expected to produce 100 Petabytes of data. This amount of data is equivalent to 10,000 times what was produced by MACHO, and approximately 200,000 modern PCs would be needed to store all that data.

After an astronomical survey is completed, an extensive analytical stage begins in order to study the information obtained. From this need for extensive analysis, a particular field of astronomy has emerged, called time domain astronomy. Time domain astronomy studies the changes of stellar objects over time. One of the many tasks in time domain astronomy is the classification of survey data into a set of known categories. These categories include many different types of stars and other astronomical phenomena of interest which can be observed, like gravitational microlensings and quasi-stellar objects. Given the amounts of data astronomers must handle, classification of survey data by means of manual inspection has become unfeasible. This problem has resulted in the use of many algorithms called

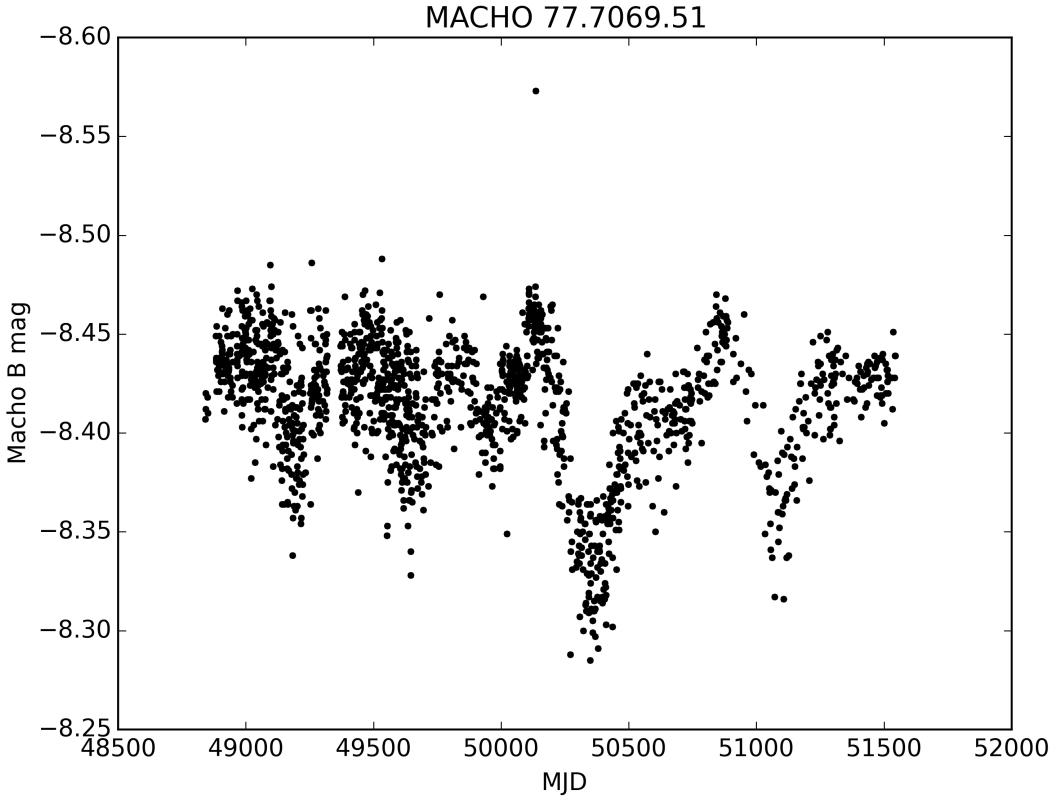

FIGURE 1.1. Lightcurve of a star. The lightcurve of an object contains the sequential measurements of its apparent brightness along different observation moments in time. Astronomers can classify a lightcurve by manual inspection, but the amount of data to be analyzed makes this unfeasible for a whole survey.

automatic classifiers which have been developed under a branch of artificial intelligence called machine learning, to analyze and then classify survey data. Automatic classifiers of survey data aren't 100% accurate, and thus the development of new algorithms and analysis techniques is still an open problem.

In time domain astronomy, data for a specific observation target comes in the form of a time series called a lightcurve. The lightcurve of a given target contains the sequential measurements of its apparent brightness along different observation moments in time. An

example of a lightcurve is shown in Figure 1.1. Lightcurves vary a lot in length, some lightcurves have as few as 100 observations while others have 1500 or more. Also, the time between observations is not constant and varies greatly, which makes many analysis tasks more difficult. The complexity of lightcurve data makes it very difficult for an automatic classifier to achieve good performance by analysing the time series data in its raw form. A much better performance is achieved when lightcurves are first transformed to a vector of many descriptors which aim to summarise the information in the time series.

Automatic classification of survey data, particularly of variable stars, has received substantial attention in the research community in the last years (Debosscher et al., 2007; Wachman et al., 2009; Kim et al., 2009; Wang et al., 2010; Richards et al., 2011; Bloom & Richards, 2011; Kim et al., 2011; Pichara et al., 2012; Bloom et al., 2012; Pichara & Protopapas, 2013; Kim et al., 2014; Nun et al., 2014; Masci et al., 2014; Hanif & Protopapas, 2015; Neff et al., 2015; Babu & Mahabal, 2015). Achieving a good performance with these classification 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 (Matter, 2007) and SkyMapper (Keller et al., 2007) are creating immense amounts of data, which makes 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 every time new surveys are coming.

Most of the automatic classification tools coming from the Machine Learning community are very effective in the sense that they can produce high accuracy results and work very fast in the classification stage (after the training phase). However, classification algorithms results are highly dependent on the way the data is represented, and a lot of effort is put in designing features to represent lightcurves. For example, Kim et al. (2011) used a classifier called 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. (2012) made an important improvement in accuracy thanks to the inclusion of new features coming from a Continuous Auto Regressive model. Huijse et al. (2012) made an improvement in periodic star classification 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.

1.2. Contribution of this Thesis

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 it 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 learn a model that can be then used to transform data

of the same kind to a new representation suitable for classification tasks. This process of transforming data from its raw form to another 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 et al. (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) follow a clustering-based method.

We build on the ideas in Coates & Ng (2012) 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 tens 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 and some cases better than classifiers trained using the traditional lightcurve representation, while the computational cost is significantly lower.

1.3. Overview of this Thesis

This thesis is based on the paper *Clustering based feature learning on variable stars* by Cristóbal Mackenzie, Karim Pichara and Pavlos Protopapas that was submitted to The Astrophysical Journal on December of 2015 and accepted for publication on February of 2016.

The remainder of this thesis is organized as follows: Chapter 2 introduces the relevant background theory, Chapter 3 gives an account of the previous work in feature design for variable stars and the field of unsupervised feature learning and Chapter 4 gives a detailed account of our methodology. Within Chapter 5, Section 5.1 presents the lightcurve catalogs and training sets used in this work. Section 5.2 discusses some implementation details, and we show our results in Section 5.3. We give a brief run-time analysis in Section 5.4. We state the conclusions of our work in Chapter 6.

2. BACKGROUND THEORY

2.1. Time Domain Astronomy

When a sky survey is completed, an extensive analytical stage follows which aims to analyze all the information obtained during the observation period. One of the tasks in this analysis step is the classification of stellar objects into one of many categories that include different kinds of stars and other interesting physical phenomena that can be captured through a telescope. What follows is a brief description of the physical phenomena behind the differences in star brightness and other physical characteristics of interest.

2.1.1. Variable Stars

Variable stars are stars that experience fluctuations in their brightness. Studying the variability in brightness of these stars is useful in many ways. First, it allows astronomers to infer other physical characteristics of the star such as radius, mass, luminosity. Also, it allows them to study stellar evolution and the distribution and size of the universe (Huijse et al., 2014). Variable stars can be divided into two main categories depending on the causes of the brightness variability: intrinsic variable stars and extrinsic variable stars. The brightness of intrinsic variable stars fluctuates because of physical changes occurring inside the star, while extrinsic variable stars show brightness fluctuations that are due to external causes not inside the star.

One example of an intrinsic variable star are Cepheid stars, which are radially pulsating supergiant stars that expand and contract periodically changing its size, temperature and

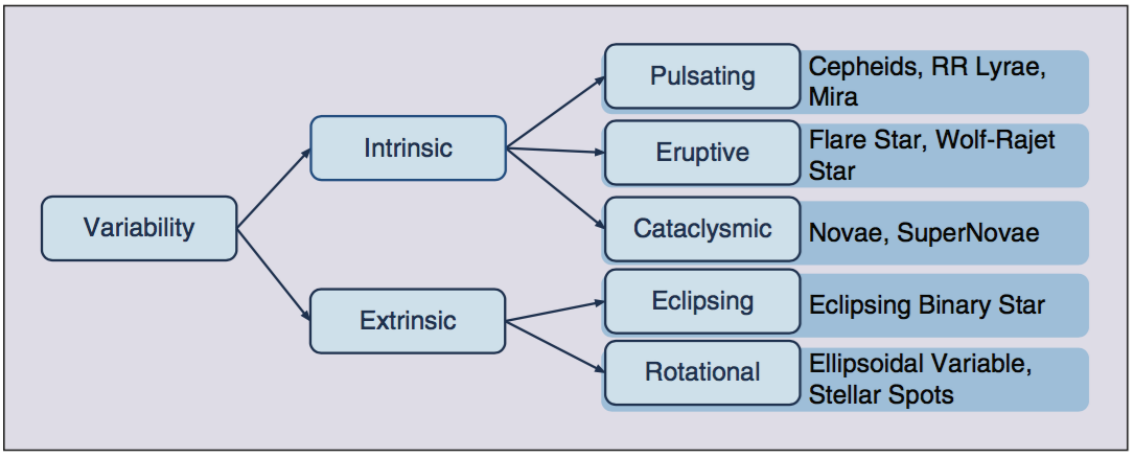

FIGURE 2.1. Variable star topological classification as presented in Huijse et al. (2014). Variable stars can be divided into two main categories depending on the causes of the brightness variability: intrinsic variable stars and extrinsic variable stars. The brightness of intrinsic variable stars fluctuates because of physical changes occurring inside the star, while extrinsic variable stars show brightness fluctuations that are due to external causes not inside the star.

brightness. One common extrinsic variable star are Eclipsing Binary stars, which are actually a system of two stars orbiting around each other with their orbital plane aligned with the earth. The periodical mutual eclipses are seen as drops in brightness in the lightcurve.

2.1.2. Variable Non-stellar Phenomena

There are other phenomena that cause brightness fluctuations in lightcurves that are not due to variable stars. A gravitational lensing effect, for example, is an increase in several orders of magnitude in the observed brightness due to a massive dark object passing in front of a light source and acting as a lens, bending the light. If the object is of planetary size the effect is called microlensing. A transiting extrasolar planet can also be detected as a brightness fluctuation: a planet outside our solar system orbiting a star with an orbital plane aligned with the earth will show up as periodic drops in the brightness of that star.

2.2. Machine Learning Background

Machine Learning is a field in computer science which aims to develop algorithms and models that can learn from and make predictions on data. Rather than following static program instructions, machine learning algorithms use models which are adapted to a set of example inputs in order to be able to make predictions on future data. The adaptation of models to a set of input data happens during an initial stage called training, where the parameters of the model are tuned with respect to an objective function which measures how well a certain model performs a given task on a set of testing data. Machine learning models can perform a variety of different tasks; some of the most popular ones are classification, regression, outlier detection, clustering and association rule extraction. The most common task is classification, which consists of predicting to which category of a given set a particular new datum belongs to, on the basis of a data set for which the categories are known. Model fitting, a process commonly known as "learning" in the machine learning community, is usually performed in one of two ways: supervised or unsupervised. What follows is a brief description of both types of learning.

2.2.1. Supervised Learning

Supervised learning refers to the process of learning a model from data where the desired output value is known. This desired output value can be a label or category, a regression value, etc. The trained model can be seen as a function that learns to map from input data to predicted output on the basis of a "training set" of $(input, output)$ pairs. Automatic classifiers commonly undergo a process of supervised learning while fitting

their models to training data. One of the most commonly used classifiers is the Support Vector Machine (Boser et al., 1992; Cortes & Vapnik, 1995). This classifier aims to find a hyperplane in the input data space that can separate training examples from two classes the best possible way through the maximisation of the margin between the hyperplane and the data at each side. New examples are classified according to which side of the hyperplane they are. Figure 2.2 shows an illustration of the Support Vector Machine classifier.

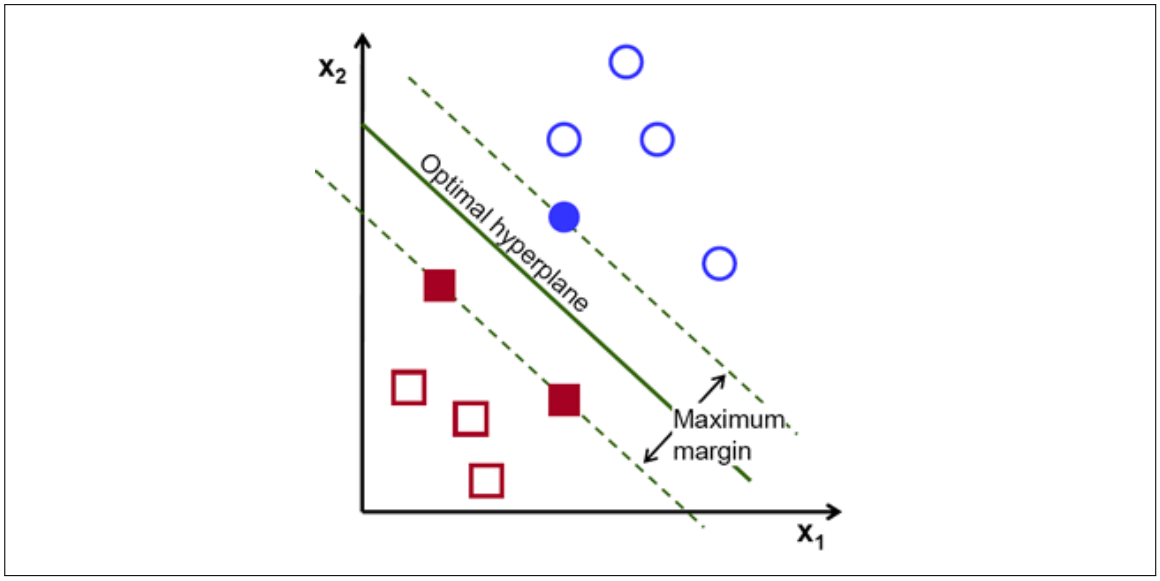

FIGURE 2.2. Support Vector Machine classifier. This classifier aims to find a hyperplane in the input data space that can separate training examples from two classes the best possible way through the maximisation of the margin between the hyperplane and the data at each side. Retrieved from `http://docs.opencv.org/2.4/_images/optimal-hyperplane.png`

2.2.2. Unsupervised Learning

Unsupervised learning is another form of model learning where the model tries to find patterns and structure data that is not associated with any desired output, called unlabeled data. The most common form of unsupervised learning are clustering algorithms, with the

most commonly widespread one being the K-Means algorithm (Hartigan, 1975; Hartigan & Wong, 1979). K-Means clustering aims to partition the n observations into k clusters where each observation belongs to the cluster with the nearest cluster centroid. Cluster centroids are first chosen at random and then are iteratively calculated as the mean of the cluster members. Figure 2.3 illustrates some steps of the K-Means learning process.

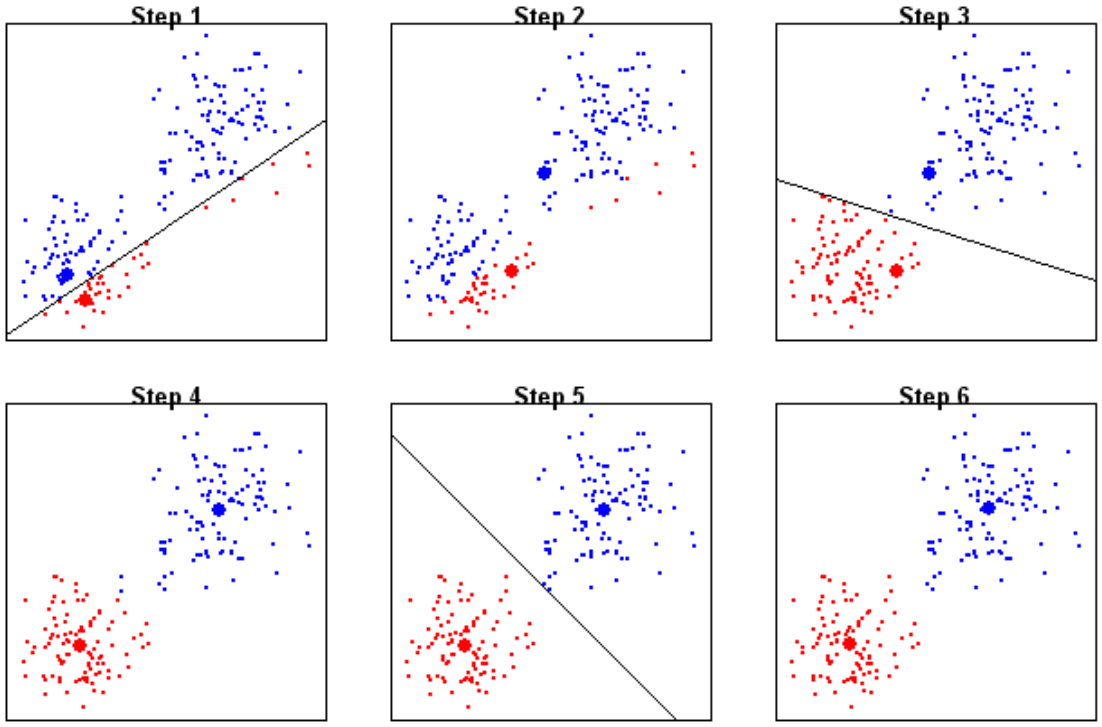

FIGURE 2.3. K-Means clustering algorithm. K-Means clustering aims to partition the n observations into k clusters where each observation belongs to the cluster with the nearest cluster centroid. Cluster centroids are first chosen at random and then are iteratively calculated as the mean of the cluster members. Retrieved from `http://astrostatistics.psu.edu/su09/lecturenotes/image/kmall.png`

2.2.3. Unsupervised Feature Learning

When input data is too complex to be successfully processed by a classifier, a pre-processing step is usually performed called feature extraction. Feature extraction is the process by which complex and often redundant data is transformed to a vector of manageable dimensionality in which each vector value contains relevant information from the input data, its goal is to reduce the amount of resources required to describe the data. This process is preceded by a first step called feature engineering or feature design, where an expert on the data's domain constructs a set of relevant values to be calculated. The main problems with feature engineering are two: expert knowledge is scarce and costly, and the designed features do not necessarily guarantee a good final model.

In recent years, an alternative to feature design has emerged in the machine learning community called unsupervised feature learning. The objective of unsupervised feature learning is to try and learn a model that can be then used to transform data of the same kind to a new representation suitable for learning tasks. This process of transforming data from its raw form to another is known as encoding. This way, the effort of finding the non-redundant characteristics of the data which can be used as features is delegated to a model which can learn complex relationships from vast amounts of data.

Unsupervised feature learning models vary a lot in nature, one of the most basic models is based on neural networks, called the Autoencoder. The Autoencoder is a neural network which tries to model the identity function of the data: the parameters of the network are iteratively adjusted so as to make the output of the network as close to the input as possible.

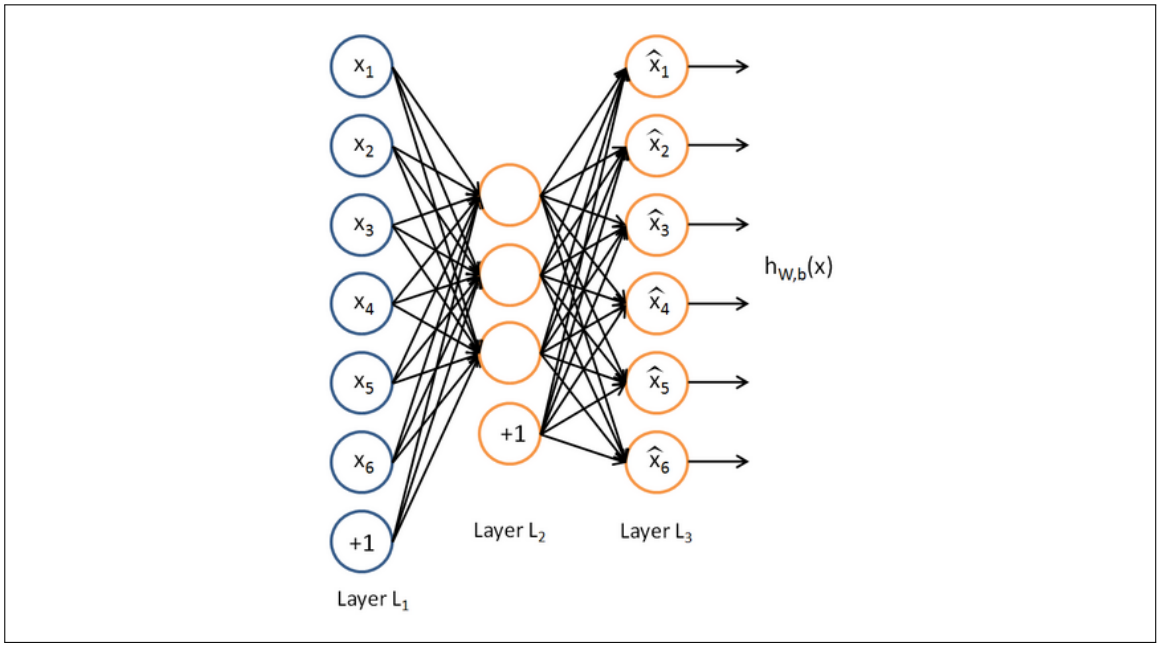

FIGURE 2.4. Autoencoder network structure. The Autoencoder is a neural network which tries to model the identity function of the data: the parameters of the network are iteratively adjusted so as to make the output of the network as close to the input as possible. Retrieved from `http://ufldl.stanford.edu/wiki/images/thumb/f/f9/Autoencoder636.png/400px-Autoencoder636.png`

Figure 2.4 shows a typical Autoencoder structure, where the outer layer of the network has the same dimensions as the input layer. The weights between the nodes are used to encode data to its feature representation.

2.3. Theoretical Foundations for our Method

Unsupervised feature learning algorithms work by learning a model from the usually vast amounts of unlabeled data available which 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 the 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 length have been proposed (Berndt & Clifford, 1994; Chen et al., 2005). Serrà & Arcos (2014) have found the Time Warp Edit Distance (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 length with even or uneven sampling, we use the Time Warp Edit Distance as the similarity measure for lightcurves in our experiments. The Time Warp Edit Distance is based on the Levenshtein Distance (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 Time Warp Edit Distance 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 apriori specification of the number of clusters to find, and it defines a representative exemplar of each clusters. 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 Time Warp Edit Distance, and lastly a detailed description of the Affinity Propagation clustering algorithm.

2.3.1. Edit Distance for Time Series

The Levenshtein Distance (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 Levenshtein Distance (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: $U = \{X_1^p | p \in \mathbb{N}\}$, X_1^p is a time series with discrete time index between 1 and p. Let x_i be the i-th 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 i-th through the j-th 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 subset x and y). We call $x \to y$ a match operation if $x \neq \Lambda$ and $y \neq \Lambda$, a delete operation if $y = \Lambda$, 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 which assigns a nonnegative 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 $i - th$ observation with m_{x_i} being its photometric magnitude and t_{x_i} the observation time.

2.3.2. Time Warp Edit Distance

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:

- *match*: The *match* operation (Figure 2.5a) 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 which represents segment (x_{i-1}, x_i) and dragging and dropping it onto the line which represents segment (y_{j-1}, y_j). The cost of this operation is proportional to the sum of the distances between corresponding samples of the segments: $|y_j - x_i|$ and $|y_{j-1} - x_{i-1}|$.
- *delete-X*: The *delete-X* operation (Figure 2.5b) consists of deleting a sample x_i. In the graphical editor paradigm, the operation consists of clicking on the point which represents sample x_i and dragging and dropping it onto the point which 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.
- *delete-Y*: Just like the previous operation, *delete-Y* operation (Figure 2.5c) consists of deleting a sample y_i. In the graphical editor paradigm, the operation

consists of clicking on the point which represents sample y_i and dragging and dropping it onto the point which 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 2.5 following the idea of edit operations in a graphical editor paradigm.

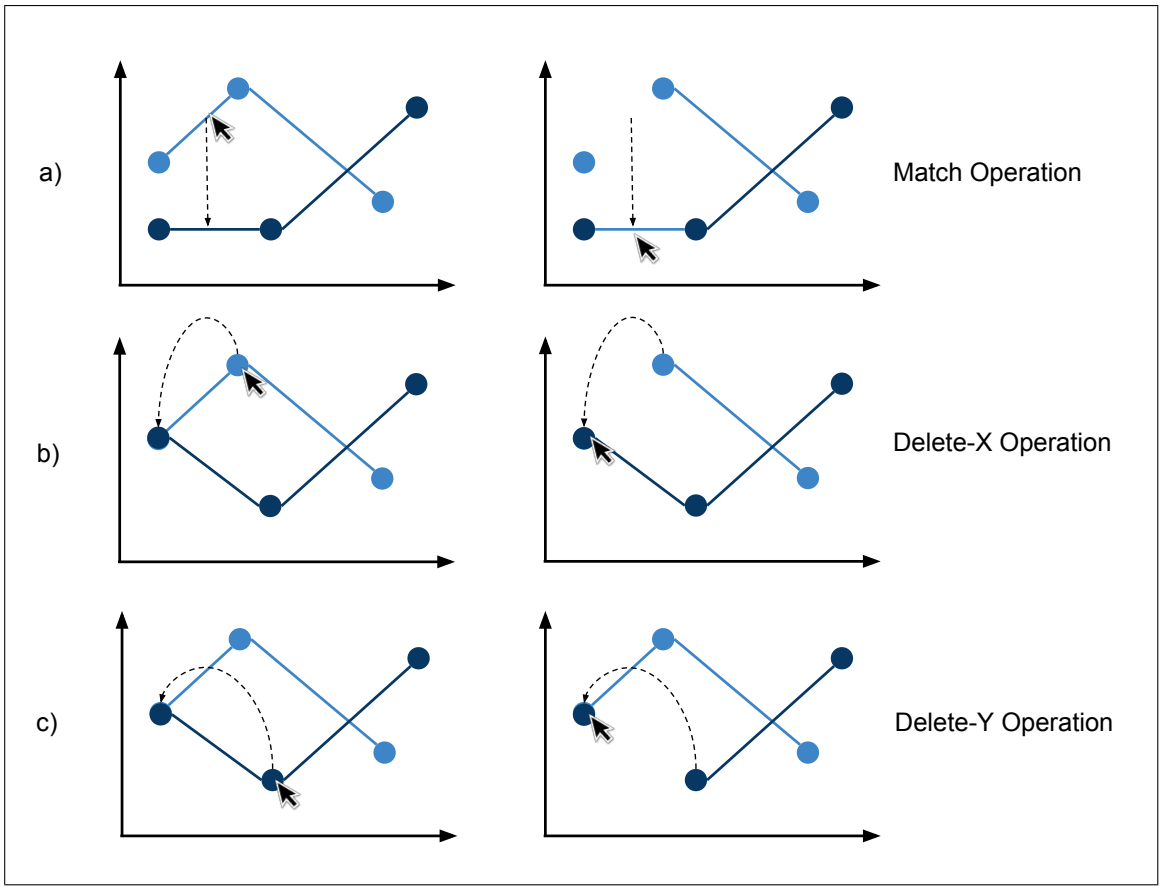

FIGURE 2.5. Edit operations in a graphical editor. Time series X and Y are depicted in light blue and dark blue, respectively.

The previous operations together with the definitions of section 2.3.1 provide the basis for the definition of TWED:

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

where

$$\Gamma_{\mathbf{x}} = |m_{x_p} - m_{x_{p-1}}| + \gamma|t_{x_p} - t_{x_{p-1}}| + \lambda$$

$$\Gamma_{\mathbf{xy}} = |m_{x_p} - m_{y_q}| + \gamma|t_{x_p} - t_{y_q}| + |m_{x_{p-1}} - m_{y_{q-1}}| + \gamma|t_{x_{p-1}} - t_{y_{q-1}}|$$

$$\Gamma_{\mathbf{y}} = |m_{y_q} - m_{y_{q-1}}| + \gamma|t_{y_q} - t_{y_{q-1}}| + \lambda$$

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 time series similarity measures, 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:

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

where

$$\begin{aligned} \Gamma_{\mathbf{x}} &= |m_{x_3} - m_{x_2}| + \gamma |t_{x_3} - t_{x_2}| + \lambda \\ &= |8 - 6| + 1 \times |8 - 3| + 1 \\ &= 8 \end{aligned}$$

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

$$\begin{aligned} \Gamma_{\mathbf{y}} &= |m_{y_3} - m_{y_2}| + \gamma |t_{y_3} - t_{y_2}| + \lambda \\ &= |7 - 8| + 1 \times |9 - 5| + 1 \\ &= 6 \end{aligned}$$

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 $\delta_{\lambda,\gamma}(X_1^2, Y_1^2)$:

$$\delta_{\lambda,\gamma}(X_1^2, Y_1^2) = \min \begin{cases} \delta_{\lambda,\gamma}(X_1^1, Y_1^2) + \Gamma_{\mathbf{x}} & \text{del} - X \\ \delta_{\lambda,\gamma}(X_1^1, Y_1^1) + \Gamma_{\mathbf{xy}} & \text{match} \\ \delta_{\lambda,\gamma}(X_1^2, Y_1^1) + \Gamma_{\mathbf{y}} & \text{del} - Y \end{cases}$$

$$= \min \begin{cases} \infty \\ 0 + \Gamma_{\mathbf{xy}} \\ \infty \end{cases}$$

$$\begin{aligned} \Gamma_{\mathbf{xy}} &= |m_{x_2} - m_{y_2}| + \gamma |t_{x_2} - t_{y_2}| \\ &\quad + |m_{x_1} - m_{y_1}| + \gamma |t_{x_1} - t_{y_1}| \\ &= |6 - 8| + 1 \times |3 - 5| \\ &\quad + |3 - 1| + 1 \times |1 - 2| \\ &= 7 \end{aligned}$$

so

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

$$= \min \begin{cases} \infty \\ \infty \\ \delta_{\lambda,\gamma}(X_1^2, Y_1^2) + 6 \end{cases}$$

$$= 13$$

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

$$= \min \begin{cases} \delta_{\lambda,\gamma}(X_1^2, Y_1^2) + 8 \\ \infty \\ \infty \end{cases}$$

$$= 15$$

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 $\Gamma_{\mathbf{xy}}$ 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.

2.3.3. Affinity Propagation

Affinity Propagation (Frey & Dueck, 2007) is a clustering algorithm which 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 indexes 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 favoured 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 flexitility is that in contrast to other common clustering algorithms like K-Means, Affinity Propagation doesn't 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 wether 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 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.

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.

3. 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, amongst others. Debosscher et al. (2007) represented lighcurves 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. (2012) proposed the use of the continuous auto-regressive 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 astromonical lightcurves which includes a compendium of features utilised 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 towards 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 et al., 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 datasets. Sparse Coding (Olshausen et al., 1996; Lee et al., 2006), a methodology that aims to learn a set of over-complete basis which 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 (Hinton & Salakhutdinov, 2006; Hinton et al., 2006; Larochelle & Bengio, 2008). The Restricted Boltzmann Machine (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 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 (Mohamed et al., 2012; G. E. Dahl 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 (Ranzato & Hinton, 2010; Krizhevsky et al., 2010) also have been used to improve on audio classification benchmarks (G. Dahl et al., 2010).

Other somewhat less popular unsupervised feature learning models that have been used with success in time series problems are the Recurrent Neural Network (Hüsken & Stagge, 2003), the Autoencoder (Poultney et al., 2006; Hinton & Salakhutdinov, 2006; Bengio, 2009) and clustering approaches (Coates & Ng, 2012). The Recurrent Neural Network (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 Autoencoder (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 is transformed into a new representation as a function of both the data and the most common data patterns found during clustering. Nam (2012) employ 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 pre-processing which requires both computational time and domain expertise. Raw time series data has 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 number of observations for a fixed time frame. These characteristics

of the data make it 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 pre-processing is a priority.

4. METHOD DESCRIPTION

Our method draws from what was proposed in Coates & Ng (2012) 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 Time Warp Edit Distance (section 2.3.2) together with an appropriate clustering algorithm that works well with any similarity measure for its data, Affinity Propagation (section 2.3.3).

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 Time Warp Edit Distance similarity measure, both described in detail in sections 2.3.3 and 2.3.2 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 4.4 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 dataset 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 4.1.

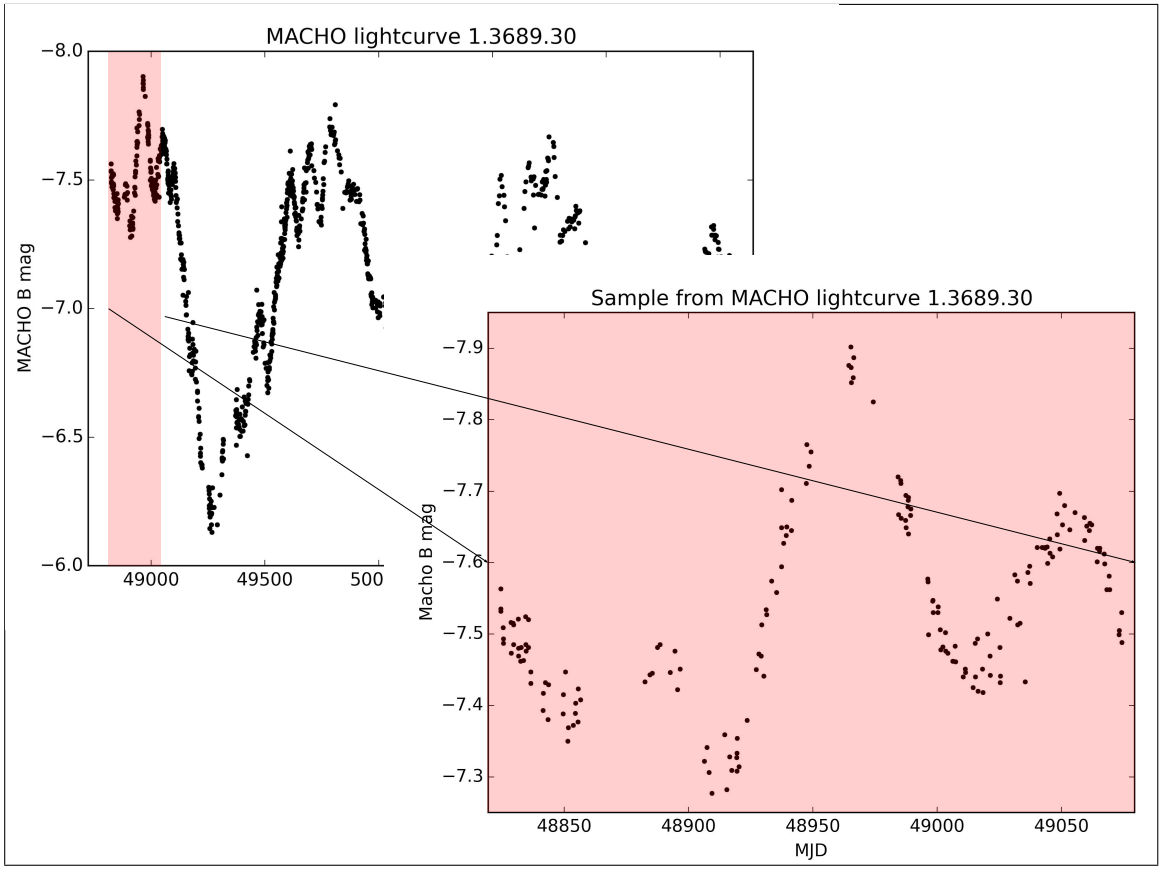

FIGURE 4.1. Lightcurve subsequence sampling. We sample a subsequence of the lightcurve by extracting all the observations in a given time window (translucent red), t_w.

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 4.2. 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 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 k-th exemplar, $\mu(\delta_{\lambda,\gamma})$ 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 the average. This value is larger

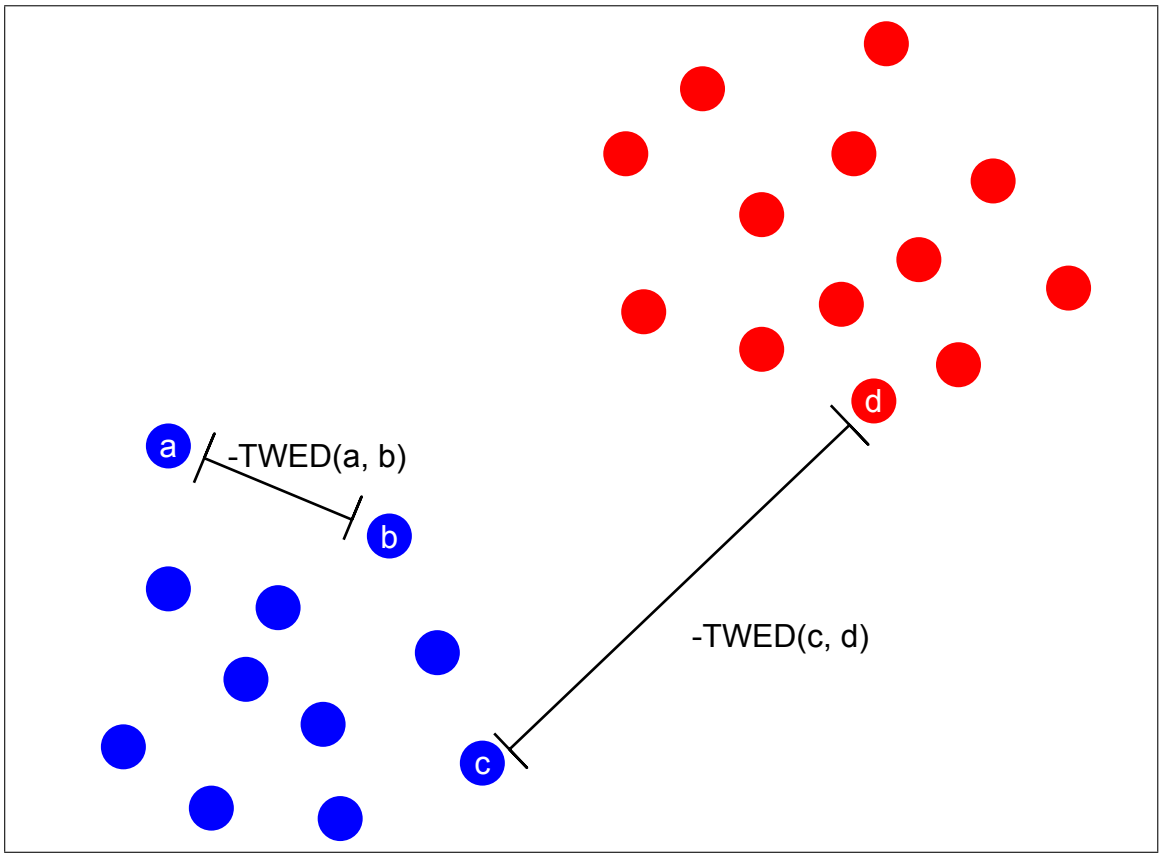

FIGURE 4.2. Lightcurve clustering. The lightcurve subsequences (represented by the colored dots) are grouped into clusters according to their affinity measure, which in this case is the negative TWED.

when the 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 t_s is usually much smaller than t_w,

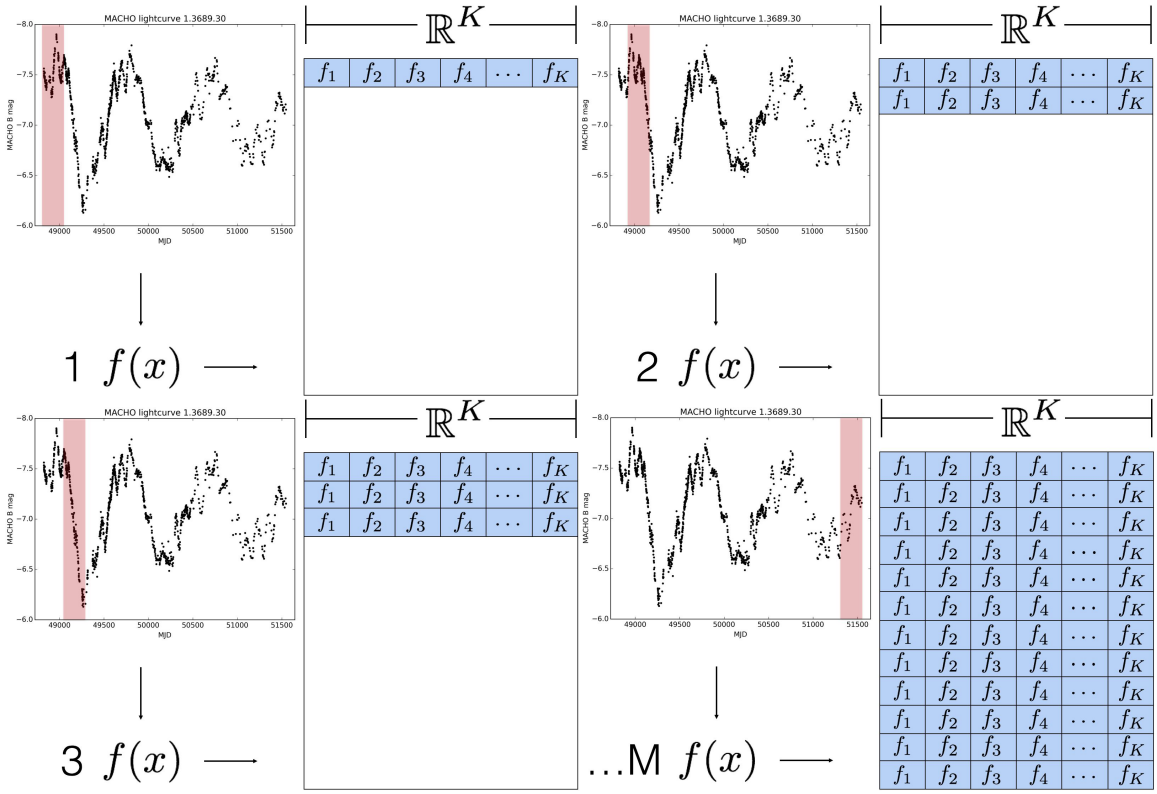

FIGURE 4.3. Sliding window process. The sliding window (translucent red) extracts a subsequence of the lightcurve at each step, which is encoded as a K-dimensional vector by our encoding function f. The window moves sequentially along the time-axis, extracting and encoding one subsequence at each step.

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 4.3.

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 equal sized regions 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 $M \times K$, which 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 4 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 dataset to train a linear Support Vector Machine (SVM) classifier (Boser et al., 1992; Cortes & Vapnik, 1995). The Support Vector Machine 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.

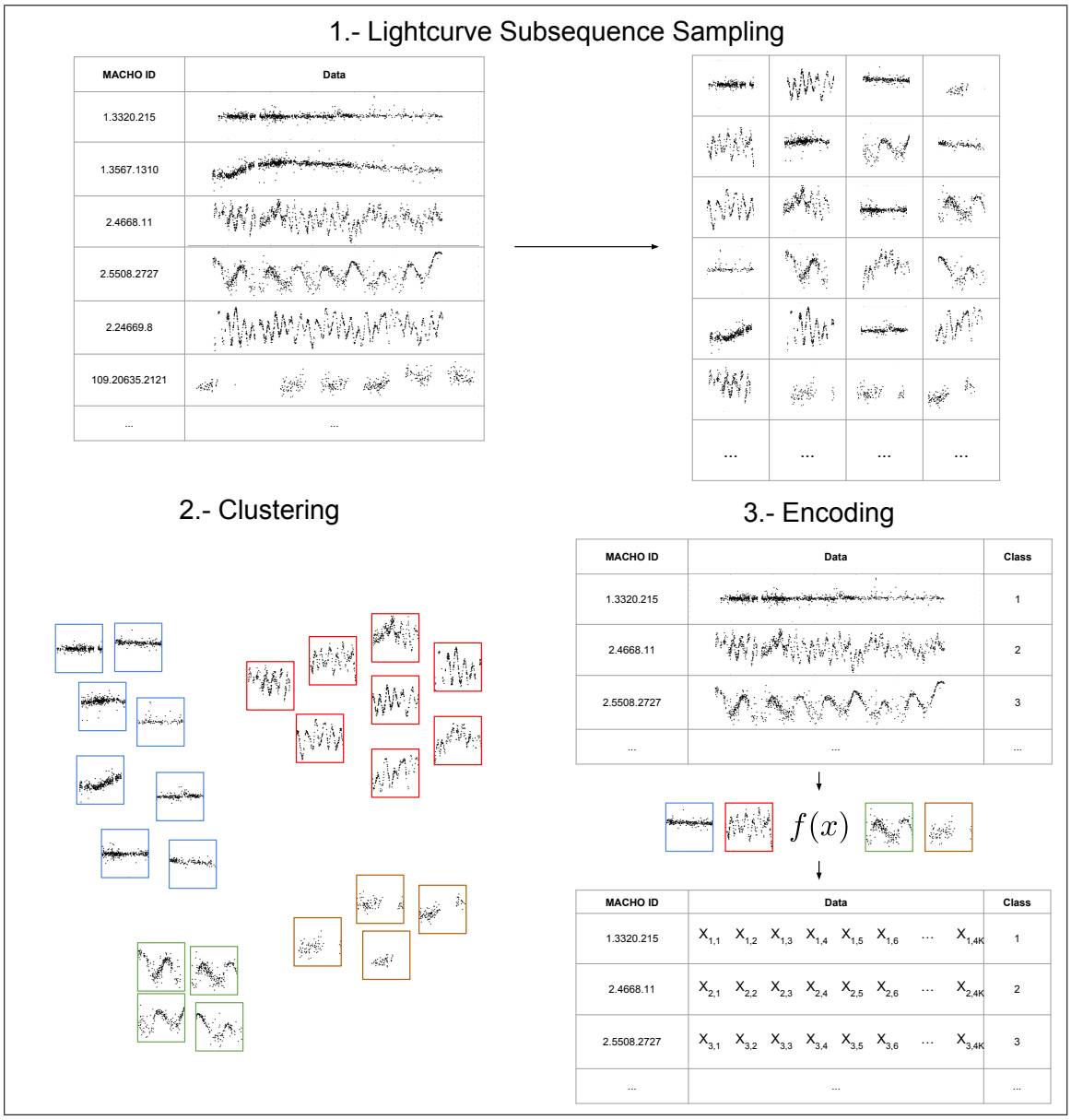

FIGURE 4.4. Method overview illustration: In the first step, we draw random subsequences from lightcurves to form a large set of lightcurve fragments. The second step consists of clustering these fragments with the Affinity Propagation algorithm. The third step consists of using the representative exemplars found during clustering to encode a training set of labeled lightcurves to a new representation more suitable for automatic classification tasks.

5. EXPERIMENTAL RESULTS AND ANALYSIS

5.1. Data

The photometric data used in our experiments belongs to two different catalogs, MACHO and OGLE.

5.1.1. MACHO Catalog

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

5.1.2. OGLE-III Catalog of Variable Stars

The Optical Gravitational Lensing Experiment (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 in the time scale of years. A by-product of these observations is an enourmous 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.

5.1.3. 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 5.1 and 5.2. 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 dataset.

TABLE 5.1. MACHO Training Set Composition

	Class	Number of Objects
1	Non Variable	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 5.2. OGLE-III Training Set Composition.

	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

5.2. Implementation

Our implementation uses minimal pre-processing: 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 datasets, the photometric errors are almost uniform across all measurements with the exception of few outliers that we remove. This means 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 5.3. The code for our experiments is available at `https://github.com/cmackenziek/tsfl`.

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

TABLE 5.3. 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 be also captured (see Figure 5.1). We also considered using the Autocorrelation function length (Kim et al., 2011) but the values of this feature for each class were too different 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 significantly the performance of the clustering step. We minimised 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 maintaining computational time within reasonable bounds and allowing us to maintain our classification performance.
TWED Elasticity Cost	γ	1e-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 towards 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 4 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, which is the most common number used throughout the literature (Boureau et al., 2010; Coates & Ng, 2012). Empirically, we found 4 to work better in the classification stage.

5.3. 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 a qualitative intuition of the inner workings of our algorithm. Then, we present the classification results on all the training sets described in section 5.1 using two different classifiers and two methods of lightcurve representation: the classical expert-designed time series features and our learned features. Finally, we present an analysis of the classification relevance in terms of both types of features.

5.3.1. 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 5.1: 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 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 previous fact explains how the final encoding step will output relevant data that allows classifiers to

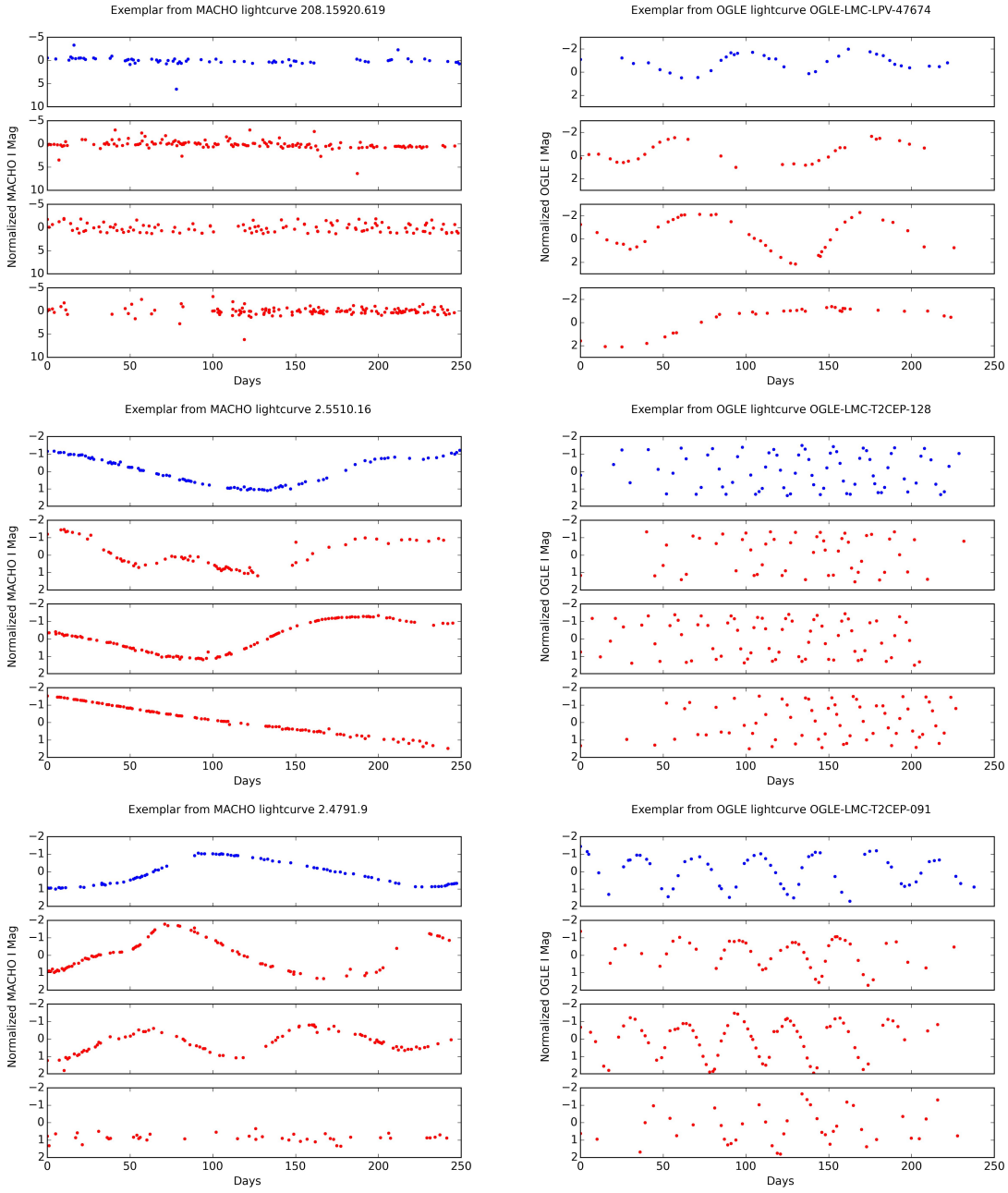

FIGURE 5.1. Cluster exemplars with members. Exemplars are lightcurve subsequences chosen by the clustering algorithm as the best representatives of their clusters. Each of the six plots shows an exemplar (plotted in blue) together with three other cluster members (plotted in red). We can appreciate how the clustering algorithm successfully groups similar lightcurve subsequences together.

TABLE 5.4. Classification F-Score on the MACHO training set.

	Class	SVM with LF	RF with TSF	SVM 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 5.5. Classification F-Score on the OGLE-III training set.

	Class	SVM with LF	RF with TSF	SVM 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

distinguish correctly between lightcurves 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.

5.3.2. Training Set Classification Results

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 is high dimensional and relatively sparse (each vector will have many zeroes by design, because of our encoding function f), we use a Support Vector Machine (Cortes & Vapnik, 1995) with a linear kernel as the classifier. To build the time series features training sets, 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 datasets 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 with the time series features. We also compared our SVM trained on learned features against an SVM trained on time series features. Tables 5.4 and 5.5 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 SVM trained on learned features achieved superior results, only performing worst 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.

5.3.3. MACHO Field 77 Classification Results

TABLE 5.6. 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

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 on Table 5.6. 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 5.2 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.

5.3.4. 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 dataset contribute 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 can be found in (Guyon et al., 2002) who used SVM coefficients for gene subset selection.

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 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 5.3 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 don 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 of Figure 5.3, where it is evident that a great percentage of the time series features don't contribute much to classification.

Another way of analyzing the contribution of learned features to classification is looking at each of the features relative importance 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 5.4 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).

5.4. 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 5.7 shows approximate the run times for each of our method steps described in Chapter 4. The two main algorithms on which we rely are

Affinity Propagation and the Time Warp Edit Distance. 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 number of samples in each of the time series under comparison.

TABLE 5.7. Computational run time details.

Step	Run Time
Sampling (Sec. 4.1)	5 minutes
Clustering (Sec. 4.2)	10 minutes
Encoding (Sec. 4.3)	1.5 - 3 hours
Total	1.75 - 3.25 hours

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 we used for comparison in our experiments, we find that the speed gain is almost an order of magnitude. 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 doesn't increase with the number of lightcurves to be transformed.

TABLE 5.8. Average encoding time.

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

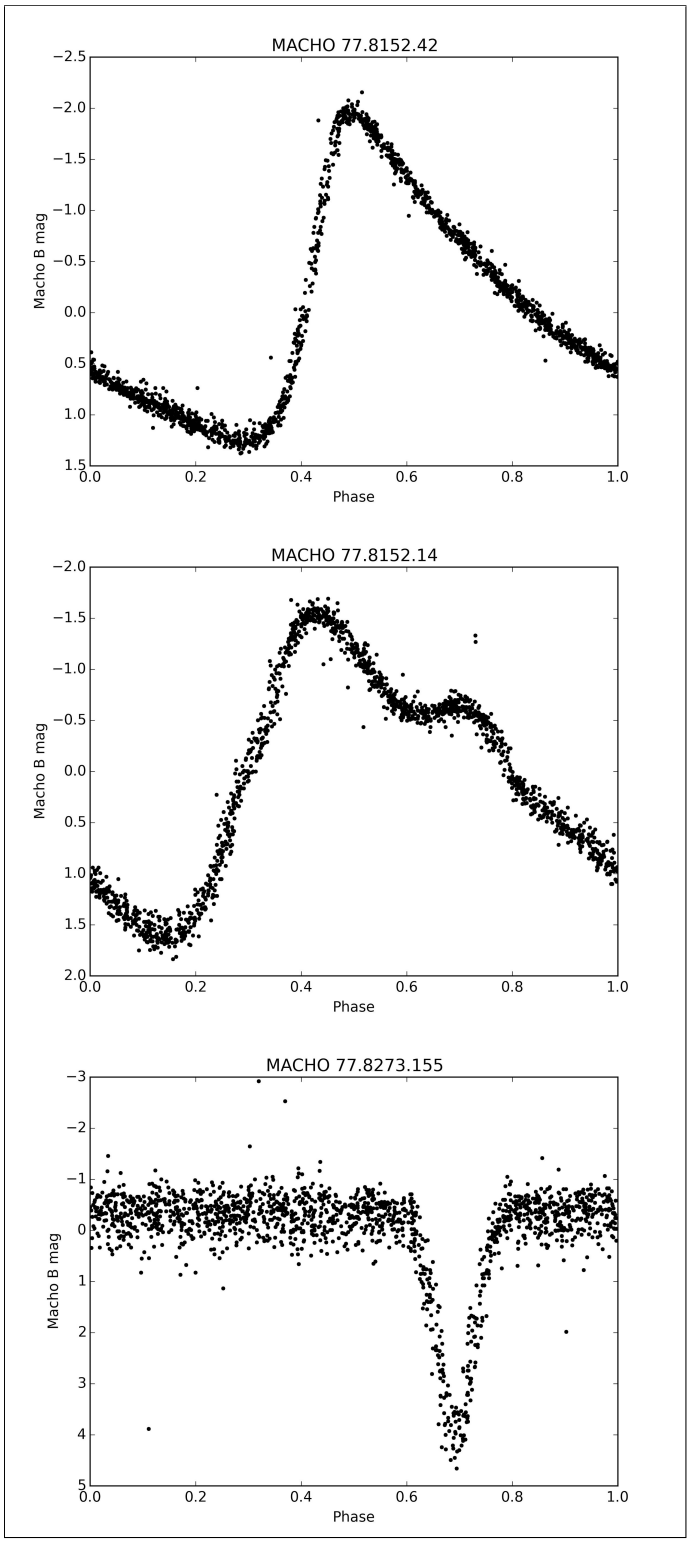

FIGURE 5.2. New variable star candidate examples. The lightcurves in the plots have been folded since they correspond to periodic stars. The first two lightcurves were classified as Cepheid while the third was classified as an Eclipsing Binary.

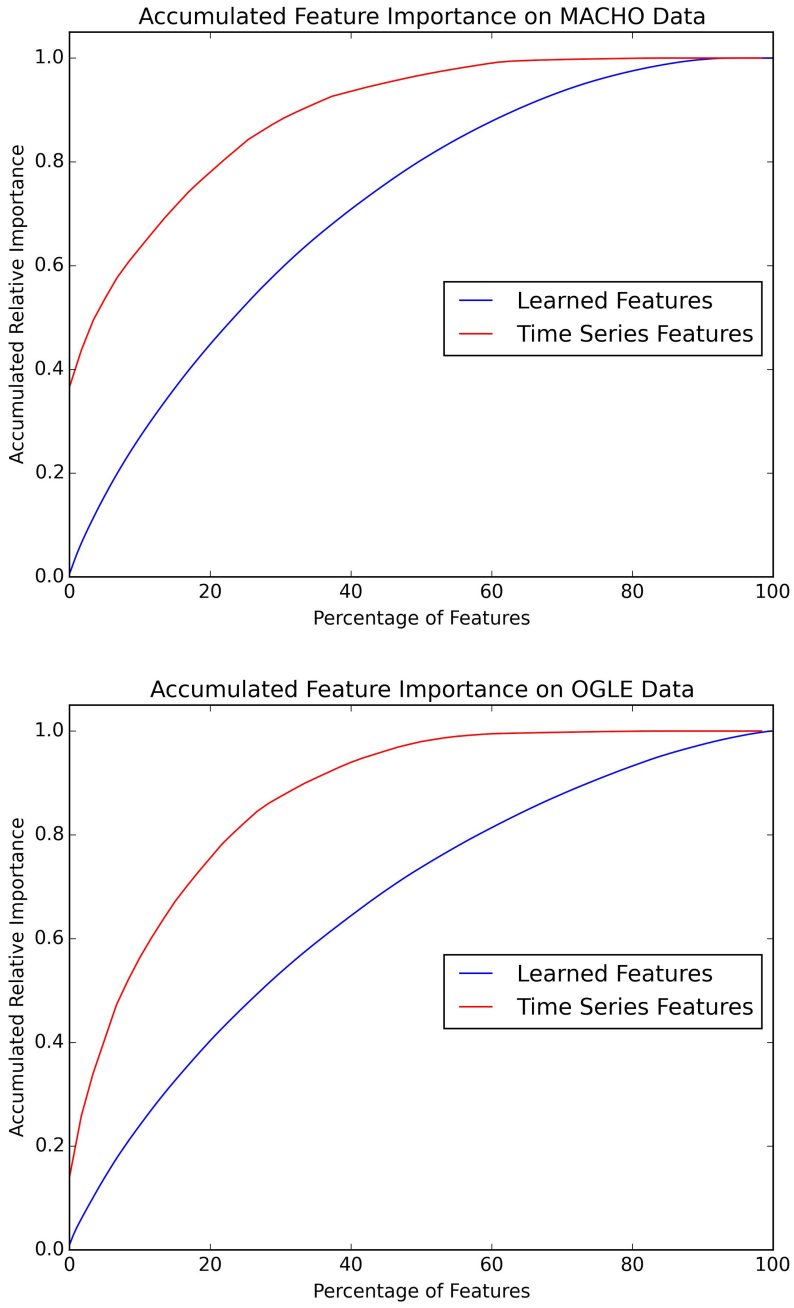

FIGURE 5.3. Relative importance cumulative sum. A feature set which has lots of features that don't contribute much to classification (i.e. rarely get used by the SVM to separate between classes) would result in a plot where the cumulative sum reaches 1 with a lower percentage of features than a feature set where most of the features are relevant to the classification task for at least some of the classes.

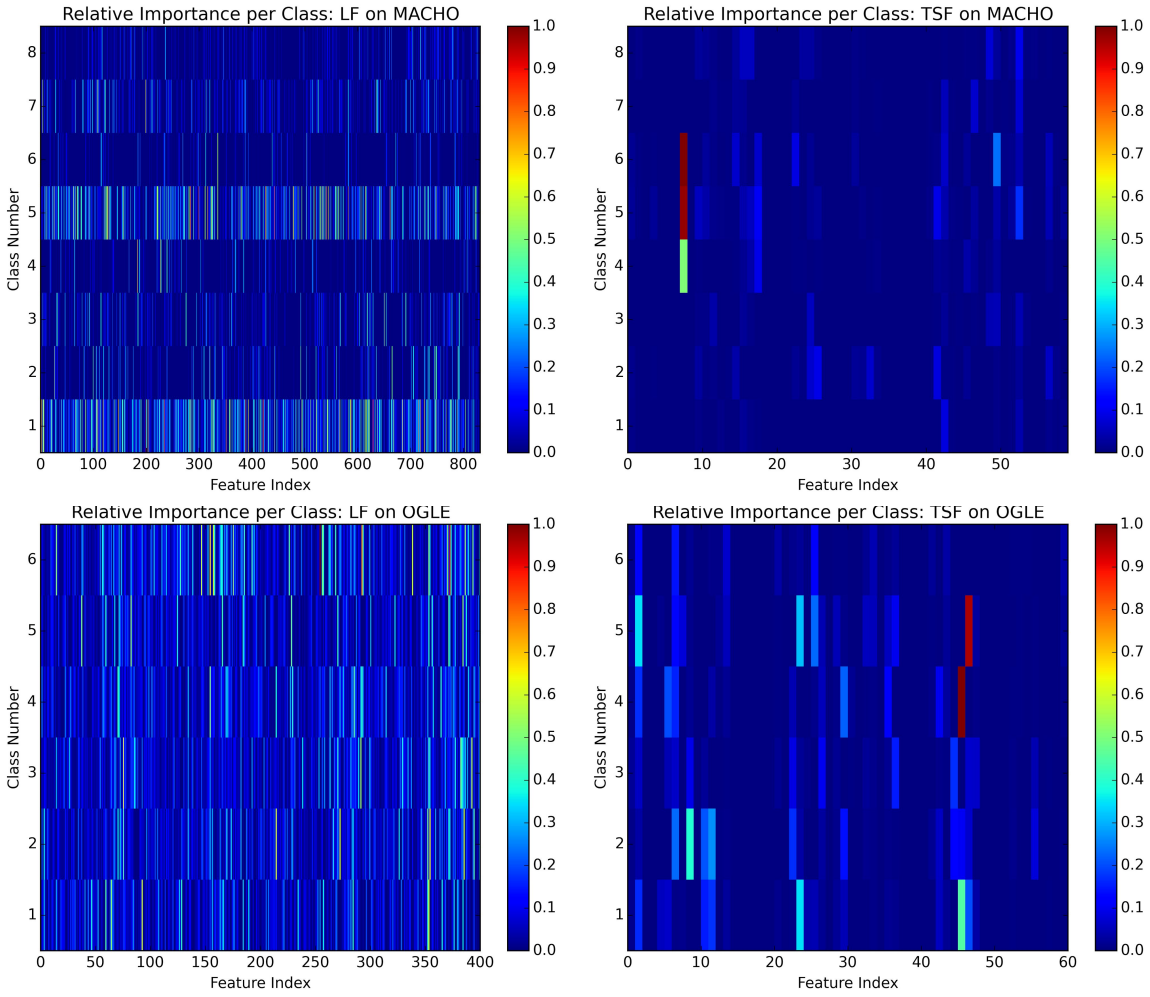

FIGURE 5.4. Relative importance per class. The two heatmaps show the relative importance of each feature of both training sets constructed with the MACHO and OGLE data. We can see in the figure that the contribution of each learned feature to classification is complex while the contribution of designed features is in many cases minimal and classification with these features is largely based on the contribution of a few of the best features.

6. CONCLUSIONS

The imminent data deluge in astronomy poses scalability challenges for data analysis. The research community must develop scalable and flexible algorithms which prescind from manual intervention in the face of new input data. One task for which this is particularly pressing is the classification of observed targets in astronomical surveys, since the amount of data to be classified has made manual inspection an infeasible solution. Automatic classifiers are the solution to this challenge, but good performance of these algorithms requires first representing lightcurves as a vector of suitable dimensionality. Lightcurves have traditionally been represented as a vector of statistical descriptors designed by astronomers called features, but these descriptors are slow to calculate, demand too much research effort to develop and do not guarantee a good classification performance.

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. This fact together with the possibility of leveraging the vast amount of information available in unlabeled data constitutes a big step towards 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 than 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 and adjust the parameters mentioned in Table 5.3 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 dataset 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 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 latter 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 that is outside the scope of this work.

Acknowledgments

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., Axelrod, T., Bennett, D., Cook, K., et al. (1997). The macho project: 45 candidate microlensing events from the first year galactic bulge data. *The Astrophysical Journal*, *479*(1), 119.

Babu, G. J., & Mahabal, A. (2015). Skysurveys, light curves and statistical challenges. *International Statistical Review*.

Bell, A. J., & Sejnowski, T. J. (1997). The independent components of natural scenes are edge filters. *Vision research*, *37*(23), 3327–3338.

Bengio, Y. (2009). Learning deep architectures for ai. *Foundations and Trends in Machine Learning*, *2*(1), 1–127.

Berndt, D. J., & Clifford, J. (1994). Using dynamic time warping to find patterns in time series. In *Kdd workshop* (Vol. 10, pp. 359–370).

Bloom, J., & Richards, J. (2011). Data mining and machine-learning in time-domain discovery & classification. *Advances in Machine Learning and Data Mining for Astronomy*.

Bloom, J., Richards, J., Nugent, P., Quimby, R., Kasliwal, M., Starr, D., et al. (2012). Automating discovery and classification of transients and variable stars in the synoptic survey era. *Publications of the Astronomical Society of the Pacific*, *124*(921), 1175–1196.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on computational learning theory* (pp. 144–152).

Boureau, Y.-L., Ponce, J., & LeCun, Y. (2010). A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th international conference on machine learning (icml-10)* (pp. 111–118).

Breiman, L. (2001). Random forests. *Machine learning*, 5–32.

Chen, L., Özsu, M. T., & Oria, V. (2005). Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 acm sigmod international conference on management of data* (pp. 491–502).

Coates, A., & Ng, A. Y. (2012). Learning feature representations with k-means. In *Neural networks: Tricks of the trade* (pp. 561–580). Springer.

Cortes, C., & Vapnik, V. (1995). Support vector networks. *Machine Learning*, *20*(3), 273–297.

Dahl, G., Mohamed, A.-r., Hinton, G. E., et al. (2010). Phone recognition with the mean-covariance restricted boltzmann machine. In *Advances in neural information processing systems* (pp. 469–477).

Dahl, G. E., Yu, D., Deng, L., & Acero, A. (2012). Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, *20*(1), 30–42.

Debosscher, J., Sarro, L., Aerts, C., Cuypers, J., Vandenbussche, B., Garrido, R., et al. (2007). Automated supervised classification of variable stars. I. Methodology. *Astronomy and Astrophysics*, *475*, 1159–1183.

Frey, B. J., & Dueck, D. (2007). Clustering by passing messages between data points. *science*, *315*(5814), 972–976.

Graves, A., Mohamed, A.-r., & Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on* (pp. 6645–6649).

Grosse, R., Raina, R., Kwong, H., & Ng, A. (2007). Shift-invariance sparse coding for audio classification. In *Proceedings of the twenty-third conference annual conference on uncertainty in artificial intelligence (uai-07)* (pp. 149–158). Corvallis, Oregon: AUAI Press.

Guyon, I., Weston, J., Barnhill, S., & Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine learning*, *46*(1-3), 389–422.

Hanif, A., & Protopapas, P. (2015). Recursive bayesian estimation of regularized and irregular quasar light curves. *Monthly Notices of the Royal Astronomical Society*, *448*(1), 390–402.

Hartigan, J. A. (1975). Clustering algorithms.

Hartigan, J. A., & Wong, M. A. (1979). Algorithm as 136: A k-means clustering algorithm. *Applied statistics*, 100–108.

Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, *18*(7), 1527–1554.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, *313*(5786), 504–507.

Huijse, P., Estévez, P., Protopapas, P., Zegers, P., Principe, J. C., et al. (2012). An information theoretic algorithm for finding periodicities in stellar light curves. *Signal Processing, IEEE Transactions on*, *60*(10), 5135–5145.

Huijse, P., Estevez, P. A., Protopapas, P., Principe, J. C., & Zegers, P.(2014). Computational intelligence challenges and applications on large-scale astronomical time series databases. *Computational Intelligence Magazine, IEEE*, *9*(3), 27–39.

Hüsken, M., & Stagge, P. (2003). Recurrent neural networks for time series classification. *Neurocomputing*, *50*, 223–235.

Jaitly, N., & Hinton, G. (2011). Learning a better representation of speech soundwaves using restricted boltzmann machines. In *Acoustics, speech and signal processing (icassp), 2011 ieee international conference on* (pp. 5884–5887).

Kaiser, N., Aussel, H., Burke, B. E., Boesgaard, H., Chambers, K., Chun, M. R., et al. (2002, December). Pan-STARRS: A Large Synoptic Survey Telescope Array. In J.˜A.˜Tyson & S.˜Wolff (Ed.), *Society of photo-optical instrumentation engineers (spie) conference series* (Vol. 4836, pp. 154–164).

Keller, S. C., Schmidt, B. P., Bessell, M. S., Conroy, P. G., Francis, P., Granlund, A., et al. (2007). The SkyMapper Telescope and The Southern Sky Survey. *Publications of the Astronomical Society of Australia*, *24*.

Keogh, E., & Lin, J. (2005). Clustering of time-series subsequences is meaningless: implications for previous and future research. *Knowledge and information systems*, *8*(2), 154–177.

Kim, D.-W., Protopapas, P., Alcock, C., Byun, Y.-I., & Bianco, F. B. (2009). Detrending time series for astronomical variability surveys. *Monthly Notices of the Royal Astronomical Society*, *397*(1), 558–568.

Kim, D.-W., Protopapas, P., Bailer-Jones, C. A., Byun, Y.-I., Chang, S.-W., Marquette, J.-B., et al. (2014). The epoch project-i. periodic variable stars in the eros-2 lmc database. *Astronomy & Astrophysics*, *566*, A43.

Kim, D.-W., Protopapas, P., Byun, Y.-I., Alcock, C., Khardon, R., & Trichas, M. (2011). Quasi-stellar Object Selection Algorithm Using Time Variability and Machine Learning: Selection of 1620 Quasi-stellar Object Candidates from MACHO Large Magellanic Cloud Database. *The Astrophysical Journal*, *735*.

Kim, D.-W., Protopapas, P., Trichas, M., Rowan-Robinson, M., Khardon, R., Alcock, C., et al. (2012). A Refined QSO Selection Method Using Diagnostics Tests: 663 QSO Candidates in the Large Magellanic Cloud. *The Astrophysical Journal*, *747*.

Krizhevsky, A., Hinton, G. E., et al. (2010). Factored 3-way restricted boltzmann machines for modeling natural images. In *International conference on artificial intelligence and statistics* (pp. 621–628).

Längkvist, M., & Loutfi, A. (2012). Not all signals are created equal: Dynamic objective auto-encoder for multivariate data. In *Nips workshop on deep learning and unsupervised feature learning.*

Larochelle, H., & Bengio, Y. (2008). Classification using discriminative restricted boltzmann machines. In *Proceedings of the 25th international conference on machine learning* (pp. 536–543).

Lee, H., Battle, A., Raina, R., & Ng, A. Y. (2006). Efficient sparse coding algorithms. In *Advances in neural information processing systems* (pp. 801–808).

Levenshtein, V. (1966). Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, *10*, 707.

Marteau, P.-F. (2009). Time warp edit distance with stiffness adjustment for time series matching. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *31*(2), 306–318.

Masci, F. J., Hoffman, D. I., Grillmair, C. J., & Cutri, R. M. (2014). Automated classification of periodic variable stars detected by the wide-field infrared survey explorer. *The Astronomical Journal*, *148*(1), 21.

Matter, D. (2007). The LSST Science Requirements Document. *Science*, 1–41.

McKinney, W. (2010). Data structures for statistical computing in python. In *Proceedings of the 9th* (Vol. 445, pp. 51–56).

Mohamed, A.-r., Dahl, G. E., & Hinton, G. (2012). Acoustic modeling using deep belief networks. *Audio, Speech, and Language Processing, IEEE Transactions on*, *20*(1), 14–22.

Nam, J. (2012). *Learning feature representations for music classification*. Unpublished doctoral dissertation, Stanford University.

Neff, J., Wells, M., Geltz, S., & Brown, A. (2015). Automated variability classification and constant stars in the kepler database. In *Cambridge workshop on cool stars, stellar systems, and the sun* (Vol. 18, pp. 879–886).

Nun, I., Pichara, K., Protopapas, P., & Kim, D.-W. (2014). Supervised detection of anomalous light curves in massive astronomical catalogs. *The Astrophysical Journal*, *793*(1), 23.

Nun, I., Protopapas, P., Sim, B., Zhu, M., Dave, R., Castro, N., et al. (2015). Fats: Feature analysis for time series. *arXiv preprint arXiv:1506.00010*.

Olshausen, B. A., et al. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, *381*(6583), 607–609.

Paczynski, B. (1986). Gravitational microlensing by the galactic halo. *The Astrophysical Journal*, *304*, 1–5.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al.(2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Pichara, K., & Protopapas, P.(2013). Automatic Classification of Variable Stars in Catalogs with Missing Data. *The Astrophysical Journal*, *777*, 83.

Pichara, K., Protopapas, P., Kim, D., Marquette, J., & Tisserand, P. (2012). An improved quasar detection method in EROS-2 and MACHO LMC datasets. *Monthly Notices of the Royal Academy Society*, *18*(September), 1–18.

Pichara, K., Protopapas, P., Kim, D.-W., Marquette, J.-B., & Tisserand, P. (2012). An improved quasar detection method in eros-2 and macho lmc data sets. *Monthly Notices of the Royal Astronomical Society*, *427*(2), 1284–1297.

Poultney, C., Chopra, S., Cun, Y. L., et al. (2006). Efficient learning of sparse representations with an energy-based model. In *Advances in neural information processing systems* (pp. 1137–1144).

Ranzato, M., & Hinton, G. E. (2010). Modeling pixel means and covariances using factorized third-order boltzmann machines. In *Computer vision and pattern recognition (cvpr),*

2010 ieee conference on (pp. 2551–2558).

Rebbapragada, U., Protopapas, P., Brodley, C. E., & Alcock, C.(2009). Finding anomalous periodic time series. *Machine learning*, *74*(3), 281–313.

Richards, J. W., Starr, D. L., Butler, N. R., Bloom, J. S., Brewer, J. M., Crellin-Quick, A., et al. (2011). On Machine-learned Classification of Variable Stars with Sparse and Noisy Time-series Data. *The Astrophysical Journal*, *733*.

Serrà, J., & Arcos, J. L. (2014). An empirical evaluation of similarity measures for time series classification. *Knowledge-Based Systems*, *67*, 305–314.

Udalski, A., Szymanski, M. K., Soszynski, I., & Poleski, R.(2008, June). The Optical Gravitational Lensing Experiment. Final Reductions of the OGLE-III Data. *Acta Astronomica*, *58*, 69–87.

Wachman, G., Khardon, R., Protopapas, P., & Alcock, C.(2009). Kernels for Periodic Time Series Arising in Astronomy. In W. Buntine, M. Grobelnik, D. Mladenic, & J. Shawe-Taylor (Eds.), *Machine learning and knowledge discovery in databases* (Vol. 5782, pp. 489–505). Springer Berlin / Heidelberg.

Walt, S. van der, Colbert, S. C., & Varoquaux, G.(2011). The numpy array: a structure for efficient numerical computation. *CoRR*, *abs/1102.1523*.

Wang, Y., Khardon, R., & Protopapas, P.(2010). Shift-Invariant Grouped Multi-task Learning for Gaussian Processes. In *Machine learning and knowledge discovery in databases* (Vol. 6323, pp. 418–434). Springer Berlin / Heidelberg.

Wenger, M., Ochsenbein, F., Egret, D., Dubois, P., Bonnarel, F., Borde, S., et al. (2000). The simbad astronomical database-the cds reference database for astronomical objects. *Astronomy and Astrophysics Supplement Series*, *143*(1), 9–22.