EmpiriciSN: Re-sampling Observed Supernova/Host Galaxy Populations Using an XD Gaussian Mixture Model

Thomas W.-S. Holoien1,2,3,4, Philip J. Marshall3,4, and Risa H. Wechsler3,4

1 Department of Astronomy, The Ohio State University, 140 West 18th Avenue, Columbus, OH 43210, USA
2 Center for Cosmology and AstroParticle Physics (CCAPP), The Ohio State University, 191 W. Woodruff Avenue, Columbus, OH 43210, USA
3 Kavli Institute for Particle Astrophysics and Cosmology, Department of Physics, Stanford University, Stanford, CA 94305, USA
4 SLAC National Accelerator Laboratory, Menlo Park, CA 94025, USA

Received 2016 December 15; revised 2017 March 6; accepted 2017 March 11; published 2017 May 11

Abstract

We describe two new open-source tools written in Python for performing extreme deconvolution Gaussian mixture modeling (XDGMM) and using a conditioned model to re-sample observed supernova and host galaxy populations. XDGMM is new program that uses Gaussian mixtures to perform density estimation of noisy data using extreme deconvolution (XD) algorithms. Additionally, it has functionality not available in other XD tools. It allows the user to select between the ASTROML and Bovy et al. fitting methods and is compatible with SCIKIT-LEARN machine learning algorithms. Most crucially, it allows the user to condition model based on the known values of a subset of parameters. This gives the user the ability to produce a tool that can predict unknown parameters based on a model that is conditioned on known values of other parameters. EmpiriciSN is an exemplary application of this functionality, which can be used to fit an XDGMM model to observed supernova/host data sets and predict likely supernova parameters using a model conditioned on observed host properties. It is primarily intended to simulate realistic supernovae for LSST data simulations based on empirical galaxy properties.

Key words: methods: statistical – supernovae: general

1. Introduction

The problem of inferring a distribution function given a set of samples from that distribution function, and the problem of finding overdensities in said distribution function are common issues in many areas of science, particularly astronomy (e.g., Skuljan et al. 1999; Hogg et al. 2005; Bovy et al. 2012). Gaussian mixture models (GMMs), which model an underlying density probability distribution function (pdf) using a sum of Gaussian components, are commonly used tools for solving density estimation problems such as these (Ivezić et al. 2014). However, traditional GMMs do not have the ability to incorporate measurement noise into the density calculation, and often, in astronomy, we must deal with observations that have multiple sources of noise with very different properties. For problems like this, the “extreme deconvolution” GMM (XDGM) technique must be used. XD was originally outlined by Bovy et al. (2011) and provides a way to perform Bayesian estimation of multivariate densities modeled as Gaussian mixtures (Ivezić et al. 2014). XDGMs have already proven useful for modeling underlying distributions using noisy observations for multiple areas of astronomy, from velocity distributions of nearby stars (Bovy et al. 2009; Bovy & Hogg 2010) to photometric redshifts and quasar probabilities of SDSS sources (Bovy et al. 2012) to 3D kinematics of stars in the Sagittarius stream (Koposov et al. 2013). However, the potential of XD models to be used as predictive tools has yet to be explored. An XDGM is able to model the complicated correlations between various parameters in a many-dimensional data set. If this model could be conditioned on the known values of some of these parameters, it could be used to predict likely values for the remaining parameters, allowing for sampling of realistic properties without knowledge of how the various parameters are correlated. This is an XD extension of the predictive approach known as Gaussian Mixture Regression, with the added functionality of being able to handle noisy or missing data. While there are multiple existing implementations of the XDGM algorithm, no existing tool has this functionality.

One potential use of an XDGM prediction tool is to sample likely parameters for a supernova (SN) given the parameters of a host galaxy. The problem of supernova simulation is a common one for large-scale sky surveys, as it is useful both for planting fake supernovae (SNe) in existing data to test detection efficiency for calculating SN rate (Melinder et al. 2008; Graur et al. 2014) and for creating realistic simulated data to test data processing pipelines of upcoming surveys, such as the Large Synoptic Survey Telescope (LSST; Ivezić et al. 2008). For various applications, it can also be useful to place realistic simulated SN within a realistic galaxy distribution in a cosmological context. For example, understanding the connection between observational biases in host detection and cosmological observables. In each of these cases, having realistic SN properties is essential for avoiding the introduction of further uncertainty or biasing detection of new sources. Furthermore, many known correlations between host and SN properties are based on physical quantities, such as host mass, metallicity, and star formation rate, which must be inferred from observations using theoretical models, introducing further uncertainty (e.g., Sullivan et al. 2010; Childress et al. 2013; Graur et al. 2016a, 2016b). An XDGM model trained only on a wide range of empirical observed host properties could be used to sample realistic supernova properties without the need for theoretical models, removing this as a source of uncertainty. However, in order to build such a tool, a new implementation of XDGM that allows for the conditioning of the model is needed. This paper is laid out as follows. In Section 2, we describe the XDGM class5 (Holoien et al. 2016a) and the new

5 The version of XDGM used in this paper is available at https://github.com/tholoien/XDGMM/tree/v1.1 and has DOI https://doi.org/10.5281/zenodo.268532.
functionality that differentiates it from existing XDGMM fitting tools. In Section 3, we describe the EmpiriciSN supernova prediction tool\(^6\) (Holoien et al. 2016b) and demonstrate its functionality. Finally, in Section 4, we summarize the capabilities of our software and describe some of the preliminary results obtained using EmpiriciSN.

2. XDGMM

As described in Section 1, XDGMM fitting methods are useful tools for performing density estimation of noisy data, a situation that occurs often in astronomy. When we began our research on building a tool to predict the properties of supernovae based on observed host galaxy properties, XDGMM modeling was a natural solution. Using machine learning to fit the underlying distributions of the numerous host and supernova properties in our data set was the right choice for our research. Furthermore, by conditioning such a model on known host properties, we can create a model based solely on the supernova properties of interest and sample from this conditioned model to predict supernova properties for a given host. Existing tools, such as ASTROML\(^7\) (Vanderplas et al. 2012; Ivezic et al. 2015) and the EXTREME-DECONVOLUTION tool\(^8\) from Bovy et al. (2011), provided XD fitting methods but did not have the ability to condition the model that we required. In addition, though the ASTROML implementation of XDGMM utilizes some of the functionality of the SCIKIT-LEARN GMM class (Pedregosa et al. 2011), neither tool implements the SCIKIT-LEARN algorithms that could be used to perform cross-validation (CV) tests to optimize model parameters. Because the ability to condition and perform cross-validation on XDGMM models could be useful for other astronomical studies, we decided to build our own implementation of XDGMM. It provides access to both the ASTROML and EXTREME-DECONVOLUTION fitting methods and implements the functionality we needed to build a supernova fitting tool and make it available to the public.

2.1. Fitting Methods

Both the ASTROML and Bovy et al. (2011) fitting algorithms are able to successfully fit a GMM using extreme deconvolution, but improving or editing their methods was not one of our goals for this project. However, as the two tools use slightly different algorithms to perform fits, we provide the ability for the user to select between the two when using XDGMM. Brief descriptions of each method are provided here.

ASTROML is an open-source Python module for machine learning and data mining and provides a wide range of statistical and machine learning tools for analyzing astronomical data sets (Vanderplas et al. 2012; Ivezic et al. 2015). One of the provided tools is an implementation of XD Gaussian Mixture Modeling and is based on the algorithms described in Bovy et al. (2011). Though slower than the Bovy et al. (2011) EXTREME-DECONVOLUTION tool, which makes use of OPENMP for parallelizing the model fitting process, the ASTROML implementation of XDGMM provides a clean user interface, similar to that of the GMM implementation of SCIKIT-LEARN (Pedregosa et al. 2011). We based our interface for XDGMM on that of the ASTROML tool. We also use the ASTROML implementation of several of the methods used to score data sets under an existing model.

The utility of extreme deconvolution for density estimation of astronomical data sets was first described in Bovy et al. (2011), and the EXTREME-DECONVOLUTION tool provided by the authors of that manuscript was one of the first tools to implement XDGMM fitting methods. Though Python, R, and IDL wrappers are available, EXTREME-DECONVOLUTION is built in C and uses OPENMP to parallelize the fitting method. As such, it provides a significantly faster fit than the ASTROML XDGMM tool.

Though the EXTREME-DECONVOLUTION provides faster performance, we elected to make the ASTROML fitting method the default method of XDGMM for two reasons. First, the ASTROML implementation provides more stable fit results and is less prone to issues resulting from outlying data than the Bovy et al. (2011) tool. Second, as a Python module, ASTROML is easily installable on most systems, while EXTREME-DECONVOLUTION requires a more detailed installation, as C compilers and the availability of OPENMP vary from system to system. Because of this, while ASTROML is required for installing XDGMM, EXTREME-DECONVOLUTION is not, and it is only imported if the user attempts to perform a fit using the Bovy et al. (2011) method.

Listing 1 shows an example of how to create a new XDGMM object, fit a model to a data set, and sample data from the model. The fitting and the sampling interface was purposefully built to mimic the ASTROML XDGMM interface so that our XDGMM class could be substituted for theirs in existing code.

Listing 1—An example of the fitting interface for XDGMM. We purposefully built this to use the same interface as the ASTROML XDGMM tool.

```python
from xdgmm import XDGMM
xd = XDGMM()
X, Xerr = (data, errors)
xd.fit(X, Xerr)
xd.sample(2000)
```

2.2. Component Selection

When fitting a GMM to a data set, it is necessary to choose the number of Gaussian components to use in the model. If the number of components in the model is too small, the model will be too simplistic and will underfit the data. However, if the number of components is too large, the model will be too flexible and will overfit the data. In either case, a subsequent sample drawn from the model will not accurately represent the data set used to train the model. In addition, with large data sets and large numbers of parameters being fit, the computation costs will rapidly become expensive, resulting in very long fit times. Thus, it is important to choose a number of components that can fit the data well without overfitting and that does not place unnecessary stress on computational resources.

2.2.1. Bayesian Information Criterion

One method for choosing the correct number of components for the model is to use the Bayesian Information Criterion (BIC; Schwarz 1978). The formula for the BIC is given in

\[ \text{BIC} = \log(L) - k \log(n) \]

where \( L \) is the maximized likelihood of the data, \( k \) is the number of model parameters, and \( n \) is the number of data points.

\(^6\) The version of EmpiriciSN used in this paper is available at https://github.com/holoien/empiriciSN/tree/v1.0 and has DOI https://doi.org/10.5281/zenodo.163859.
\(^7\) http://www.astroml.org/index.html
\(^8\) https://github.com/jobovy/extreme-deconvolution
Equation (1) below.

$$\text{BIC} = -2 \log \hat{L} + k \log n$$  

(1)

Here, $\hat{L}$ is equal to the maximized likelihood function of the model being scored, $k$ is the number of free parameters being estimated (e.g., the number of components in a GMM), and $n$ is the number of observations used to fit the model. If the BIC is calculated for a number of different models that each use a different number of free parameters, the one with the lowest BIC is the preferred model. As it incorporates both a likelihood score and a component that incorporates the number of free parameters, it penalizes models with too many degrees of freedom, which can help avoid overfitting.

Our XDOMM class computes the BIC score for the current model using a specific data set in the same way that the SCIKIT-LEARN GMM class computes the BIC. The only exception is that our class can also account for uncertainty on the input data when computing the BIC (if the user inputs a covariance matrix with a data set, the XDOMM class will incorporate these uncertainties into the model covariance matrix before calculating the BIC). We have also provided a function that can compute the BIC for a given data set for a range of numbers of components, which allows the user to compare different models and select the ideal one. This functionality is demonstrated in Listing 2.

Listing 2—A demonstration of the BIC test function, which allows the user to compare different XDOMM models with different numbers of components.

```python
param_range = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
bic, optimal_n_comp, lowest_bic = xd.bic_test(X, Xerr, param_range)
```

In the example code of Listing 2, the XDOMM object computes the BIC score for the data and uncertainties contained in the X and XERR arrays for a number of components ranging from 1 to 10. It then returns the BIC array, which contains the score for each model, the optimal number of components (defined as the number of components in the model with the lowest BIC score), and the lowest BIC score. These results can then be used for direct comparison or can be plotted to view the results visually. In Figure 1, we show the BIC results for the ASTROML XDOMM demo data set (Vanderplas et al. 2012). The optimal number of components occurs with five Gaussian components in the model, indicating that models with higher numbers of components overfit the data.

2.2.2. Machine Learning

An alternative way to determine the number of components is to perform a cross-validation test with a range of numbers of components. In order to allow the user to perform such a test, we have made the XDOMM class a subclass of the SCIKIT-LEARN BASEESTIMATOR class and implemented all of the SCIKIT-LEARN functions necessary for the standard SCIKIT-LEARN cross-validation tools. We use the mean log-likelihood of a data set under the given model as the score for cross-validation. Because XDOMM extends BASEESTIMATOR, a cross-validation test can be performed by simply passing an XDOMM object and a data set into the SCIKIT-LEARN cross-validation functions. A demonstration of computing a validation curve using the same ASTROML demo data set for 1–10 components is given in Listing 3.

It is important to note the trick used to pass errors to the SCIKIT-LEARN methods. Normally for unsupervised learning, you would only pass an X “design matrix” array, and not a “target” y array, to the VALIDATION_CURVE method. However, an error array must be passed to the XDOMM FIT method, and VALIDATION_CURVE simply uses its y input as the second argument for the FIT function. Thus, by passing in the error array as the “target” array, we can pass it to our FIT function so that it can be used in fitting the data.

Figure 2 shows the results of the cross-validation test above. The cross-validation test prefers the maximum number of components (10) that we allowed for the model (and in fact, the score continues to rise as more components are added beyond 10). This is a result of the particular data set being fit: the likelihood of the data being fit increases with more components, and there is enough structure to the data that even with a large number of Gaussians, the trained model continues to be a good predictor of new data. However, increasing the number of components in the model rapidly causes the fit algorithm to become computationally expensive, especially for the ASTROML algorithm. While a model with a large number of components may be mathematically superior, in most cases, the BIC seems to provide a way to find a model that is “good enough” to fit the data well while also maintaining the number of components at a value that keeps the computation of new fits reasonable. We recommend trying both tests with a given data set to see if the results differ substantially before settling on a choice for the number of components to use when fitting a model.
Once we know the optimal number of components to use, it is straightforward to fit a model to the data (see Listing 1). Figure 3 replicates the results of the ASTROML Extreme Deconvolution example (Vanderplas et al. 2012) using our XDGMM code. We first create a “true” distribution and a “noisy” distribution using the Vanderplas et al. (2012) demo code, then we fit a model to the noisy distribution using five Gaussian components and sample 2000 data points from the model. We can see that even with only five components, the distribution sampled from the XDGMM model is able to replicate the true data sample despite being fit using the “measured” noisy distribution. This demonstrates why extreme deconvolution is such a powerful tool for modeling data sets such as this.

2.3. Conditioning the Model

A primary motivation behind this implementation of extreme deconvolution was to develop a tool that can be used to predict model parameters based on known values of other parameters. To do this, the model must be conditioned on the known parameter values, after which we can then sample values of the non-conditioned parameters from the conditioned model. Neither the ASTROML nor the Bovy et al. (2011) implementations of XDGM contain this functionality, and we have implemented it in our software.

First, we briefly discuss the mathematics of a conditional GMM. The probability distribution for a Gaussian mixture with \( K \) components is given by (Bishop 2006; Rasmussen & Williams 2006):

\[
p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x | \mu_k, \Sigma_k).
\]

Here, \( \pi_k \), \( \mu_k \), and \( \Sigma_k \) are the mixing coefficient (weight), means, and covariances of the \( k \)th Gaussian component, respectively. Given the jointly Gaussian vectors \( x_A \) and \( x_B \) and the above Gaussian mixture, we have:

\[
x = \begin{pmatrix} x_A \\ x_B \end{pmatrix}, \quad \mu_k = \begin{pmatrix} \mu_{kA} \\ \mu_{kB} \end{pmatrix}.
\]

The conditional distribution of \( x_A \) given \( x_B \) for the \( k \)th Gaussian component is given by (Bishop 2006):

\[
p_k(x_A | x_B) = \frac{p_k(x_A, x_B)}{p_k(x_B)} = \mathcal{N}(x_A | \mu_{kA|B}, \Sigma_{kA|B}^{-1}). \tag{3}
\]

Finally, the \( k \)th conditional mixing coefficient is given by:

\[
\pi'_k = \frac{\pi_k \mathcal{N}(x_B | \mu_{kB}, \Sigma_{BB})}{\sum_{k=1}^{K} \mathcal{N}(x_B | \mu_{kB}, \Sigma_{BB})} \tag{5}
\]

Thus, the conditional probability distribution for the whole GMM is given by:

\[
p(x_A | x_B) = \sum_{k=1}^{K} \pi'_k p_k(x_A | x_B). \tag{6}
\]

The resulting conditioned GMM has the same number of Gaussian components as the original GMM, but has fewer dimensions, given by the number of dimensions in \( x_A \). We can then use this conditioned model to sample values for the parameters in \( x_A \) given the known quantities in \( x_B \).

When using XDGM as a prediction tool for astronomical quantities, it may often be the case that the user wants to condition the model on parameter measurements that have measurement uncertainties. Conditioning a model on a particular value of \( x_B = x_{B,0} \) is equivalent to marginalizing out \( x_B \) assuming a delta function PDF for it. If we include uncertainties on the measurement of \( x_B \), we can incorporate these uncertainties into the conditioning by adding \( C_B \) to the covariance array of
the unconditioned GMM prior to conditioning the model. The result will still be a GMM, but its components will be (i) broader, as the extra covariance $C_B$ will be added to the component covariance matrix $A_{ii}$, and (ii) weighted differently, as the weights in Equation (5) are themselves functions of $x_B$.

We have built two different but equally straightforward interfaces for conditioning the model. In order to condition the model, the XDGMM object must be informed which of the parameters of the model (e.g., $x$ and $y$ in the sample data set) have values on which to condition the model. The XDGMM object stores the parameters in a specific order based on their order in the data set that was used to fit the model. For example, $x$ is stored first and $y$ second for the demo data set used here, because the data was passed to the model as $(x, y)$ pairs. In some cases, such as this simple demo, it may be easy for the user to remember the order of the parameters, and we have built one conditioning interface to take advantage of such cases. Listing 8 demonstrates this first interface. In this conditioning method, the user passes one or two arrays to the CONDITION function; one array containing values for each parameter (either a value for conditioning or NAN if the parameter is not being used for conditioning) and an optional second array containing uncertainties on the parameter values.

However, we recognize that in many cases, the user may be fitting large data sets with many parameters, and maintaining the proper order for conditioning may be difficult. For this reason, the XDGMM object also allows the user to label the parameters in the model. This can be done manually, by setting the XDGMM object’s LABELS array, or by fitting data stored in a PANDAS DATAFRAME object. If the data being fit are stored in a PANDAS DATAFRAME and the columns are labeled, the labels will be automatically stored in the XDGMM object when the data are fit.

Listing 4—A demonstration of the first interface for model conditioning using the demo data set and the known value $y = 1.5 \pm 0.05$. In this method, the user knows the indices of the parameters to use for conditioning and passes arrays for the parameter values and uncertainties to the CONDITION function.

```python
fixed_X = np.array([np.nan, 1.5])
unc = np.array([0.0, 0.05])
new_xd = xd.condition(X_input = fixed_X,
Xerr_input = unc)
```

If the labels have been set, the user can then use a dictionary object that links the labels for parameters to condition the model on with (value, uncertainty) pairs. In Listing 5, we demonstrate this functionality. Here, we label the parameters “$x$” and “$y$” and then pass a dictionary containing only a value for $y$ to the CONDITION function. The conditioned model that results will be the same regardless of the method used for conditioning; the different interfaces are simply supplied so that the user can choose whichever method they prefer.

Listing 5—A demonstration of the second interface for model conditioning. Labels for the different parameters in the model are first set by the user, and then a dictionary object is used for conditioning.

```python
xd.labels = np.array(['x', 'y'])
fixed = {'y':(1.5, 0.05)}
new_xd2 = xd.condition(X_dict = fixed)
```

Once the model has been conditioned, it will have the same number of components as the original model but will have fewer dimensions, as it is now a model only for the parameters that were not conditioned out. In the code above, we have conditioned the demo model based using $y = 1.5 \pm 0.05$, and the resulting model is a five-component GMM for the $x$ parameter. Figure 4 shows the resulting probability distribution of $x$. Conditioning the model changes the weights, means, and covariances of the components of the model. The constraint on $y$ essentially rules out several components of the original model, significantly reducing their weight in the conditioned model.

As stated before, one potential use of a conditioned model is to create a “prediction engine” that can predict some parameters using an XDGMM model conditioned on known values of other parameters. To demonstrate this, we sampled 1000 data points from our original unconditioned model to create a data set to be compared with our predictions. This represents a new “observed” data set for the $x$ and $y$ parameters. If we had only observed the y-values from this data set and wanted to predict a likely $x$-value for each $y$-value, we could condition the model on each of these $y$-values in turn and draw a predicted $x$-value from the conditioned model. In reality, XDGMM would likely be used to predict values for parameters that have not been measured, so this provides a good way to test whether the tool is functioning properly—these predicted $x$-values should follow the same distribution of the observed $x$-values—and Figure 5 shows that this is the case. Although the predicted $x$ for a single given $y$-value may not match the observed $x$-value, the fact that the overall distributions match indicates that XDGMM provides an accurate prediction tool.

Figure 5 also compares the performance of our XDGM model with a standard random forest predictive method. We fit a random forest model to the same noisy data set used for the demo, and then used this model to predict $x$-values for the same $y$-values used for the XDGMM test. As the random forest model was fit on the noisy data and does not handle data uncertainties, it is unable to replicate the tight “true” distribution of our test data set. While this is a simple example...
and there are other predictive machine learning methods that could potentially model the data, this demonstrates that XDGM has capabilities, such as accounting for correlations in predicted values and handling noisy or missing data, that other methods do not.

Throughout this section we have used the ASTROML Extreme Deconvolution example data set (Vanderplas et al. 2012), because it is a simple two-dimensional data set that makes the capabilities of XDGM (e.g., fitting a model to a data set, sampling from a model, and conditioning a model) easy to visualize. Using this data set also allows us to compare our XDGM class directly with the ASTROML implementation. However, this data set is very simplistic and could likely be modeled equally well with a number of different modeling techniques. In order to provide an example of a more realistic use case for an XDGM model, Figure 6 shows the results of fitting a 12-component XDGM model to stellar data from SDSS Stripe 82 (Ivezić et al. 2007).

This data set, which is also used in the ASTROML Extreme Deconvolution examples, is a five-dimensional data set consisting of the g-band magnitude and $u - g$, $g - r$, $r - i$, and $i - z$ colors of roughly 13,000 stars from the SDSS Stripe 82 Standard Star Catalog (Ivezić et al. 2007). Magnitudes derived from single epoch observations are used as a low signal-to-noise input sample for our XDGM model, and magnitudes derived from multiple observations (and, thus, with smaller scatter) are used as a comparison sample. Figure 6 shows two dimensions, the $g - r$ and $r - i$ colors, of the five dimensions that were fit. The high signal-to-noise data is shown in the top-left panel and the low signal-to-noise input data is shown in the top-right panel. The XDGM model was fit using 12 Gaussian components and the Bovy et al. fitting method was fit with the resulting Gaussian components, shown in the bottom-right panel. The bottom-left panel shows the distribution of an equal number of points sampled from the fit model, and as the figure shows, the points sampled from the resulting distribution have scatter comparable to the Standard Star Catalog data, despite being fit to more noisy input data. This example shows that XDGM can be a powerful tool for modeling the underlying distributions of noisy observed astronomical data sets, and that our XDGM model could then be conditioned to, for example, sample likely star colors given only a g-band magnitude.

3. EmpiriciSN

It is well established that supernova properties, such as light curve color and shape, have a dependence on their host environments (e.g., Modjaz et al. 2008; Sullivan et al. 2010; Childress et al. 2013; Galbany et al. 2014; Graur et al. 2016a, 2016b). Planting simulated supernovae in real or simulated imaging data has a variety of uses—for instance, fake supernovae can be used to test the detection efficiency of supernova searches, which is a necessary quantity to know for calculating supernova rates. Simulated data, including supernovae, can be used to train data processing and automated source detection pipelines for future surveys, like LSST. Modeling the distribution of supernovae within a cosmologically motivated galaxy distribution can also be used to test how the connection between their properties and the underlying structure can impact various observables.

It is important to ensure that supernovae simulated for these purposes have properties that accurately match their
environments and are consistent with each other. Otherwise, the results of these studies would not be applicable to real-world conditions. However, many of the known supernova-host correlations are based on physical parameters of the host—e.g., star formation rate, mass, metallicity—which must be inferred using theoretical models from observations (e.g., Sullivan et al. 2010; Childress et al. 2013; Graur et al. 2016a, 2016b). Although many of these theories are fairly well established, this introduces additional uncertainty into the selection of supernova properties.

In this section, we outline EmpiriciSN, a tool for predicting realistic Type Ia supernova (SN Ia) properties based on observed host galaxy properties. Our goal in creating EmpiriciSN was to provide a model trained on observed empirical host and supernova properties, thus, eliminating the need for using theoretical models to infer the host galaxy’s physical properties. As this requires calculating correlations between many supernova and host properties and the subsequent conditioning of the model, it provides a real-world use for our XDGMM class. Our default model is trained using supernova and galaxy properties that can be generated by SNCOSMO (Barbary 2014) and CATSIM (Connolly et al. 2014), so that it can be used for generating realistic supernovae for LSST Twinkles® simulations (LSST DESC 2017, in preparation). These include the SALT2 Type Ia light curve parameters (X0, X1, and c; Guy et al. 2007), the host redshift, the 10 rest-frame host colors obtainable with ugriz magnitudes, the separation of the supernova from the host nucleus in units of the host effective radius, and the local surface brightness at the location of the supernova in all five ugriz filters. These host properties were chosen because they are all observable properties that do not require theoretical modeling and are known to correlate with physical properties of the host environment that can affect supernova parameters.

3.1. Input Catalogs

In order to model the 20 host and supernova properties listed above, we require a large data set with consistent measurements of the SALT2 parameters and consistent host photometry. We decided to use a sample of supernovae taken from the Supernova Legacy Survey (SNLS; Astier et al. 2006; Sullivan et al. 2011) and the Sloan Digital Sky Survey (SDSS; York et al. 2000) Supernova Survey to build a model for our data. All SALT2 light curve parameters for the SNLS supernovae and a portion of the parameters for our SDSS supernovae are taken from the Joint Light-curve Analysis (JLA; Betoule et al. 2014), a project to analyze light curves of supernovae discovered by SNLS, SDSS, and other sources. This includes 242 spectroscopically confirmed Type Ia supernovae from SNLS and 374 spectroscopically confirmed Type Ia supernovae from the SDSS SN survey. Because the JLA catalog provides a peak magnitude rather than the X0 SALT2 parameter, we used SNCOSMO to fit the JLA light curves ourselves, assuming the redshifts, X1, and c parameters provided, and calculated X0 in this way. In order to increase the size of our sample to be large enough to model, we also include the remaining spectroscopically confirmed SNe Ia from the SDSS SN search as well as the photometric SNe Ia with a spectroscopic host redshift from Sako et al. (2014). We include the photometric SNe Ia, because doing so provides us with an additional 906 SNe, though we acknowledge that some of these may not actually be Type Ia. Sako et al. (2014) provide the X0 SALT2 parameter, but as they use a slightly different model than the JLA model, we correct the Sako et al. (2014) X0 values by a factor of $10^{0.108}$ to make them consistent with the values measured from the JLA light curves.

We then searched the coordinates of all host galaxies from our SN samples in the SDSS Data Release 12 (DR12; Alam et al. 2015) and obtained model magnitudes, model fluxes, effective radii, and morphology for all galaxies that were detected in DR12 data. This reduced our sample to 159 supernovae from SNLS and 1273 supernovae from SDSS. For the purposes of calculating local surface brightness, all galaxies in the sample are considered to have either an exponential or a de Vaucouleurs surface brightness profile, and we use whichever model is considered more likely by the SDSS pipeline in the r-band as the model for each galaxy. The r-band effective radius of the best-fit surface brightness profile was used to convert the separation from the host nucleus measured in Betoule et al. (2014) and Sako et al. (2014) from arcseconds to units of $R/R_*.$

The host galaxy model magnitudes are magnitudes generated for each filter, assuming the same best-fit surface brightness profile, and they incorporate a convolution with the image PSF to account for PSF effects. The magnitudes have been corrected for Galactic extinction and corrected to rest-frame using KCORRECT (Blanton & Roweis 2007) before being used to calculate host colors. The model magnitudes are ideal for unbiased galaxy color measurements, and while other data sets may not measure galaxy magnitudes and colors in the same way, we believe they should still be able to obtain reasonable results from our default model.

Although we have taken steps to make our SN and host galaxy sample as consistent as possible, we acknowledge that both SN data sets were obtained after applying selection criteria, particularly the JLA sample, and we have not incorporated any corrections into our model to correct for these selection effects. Thus, the distribution probability of SN parameters obtained by EmpiriciSN combines both the true underlying distribution in nature and the detection efficiencies of each survey, and it may be necessary for the user to correct for the detection efficiency when using our default model to simulate SNe. We also caution that, though the data set spans a wide range of SN and host galaxy properties, it is possible that certain galaxy types (e.g., low-mass galaxies) are under-represented and, therefore, not well-modeled by the default model. We will be taking steps to quantify potential shortcomings of the default catalog in a future release of the software.

However, despite the survey effects present in our default data sets, when combined, they still provide us with a fairly large data set with a wide range of SN and host galaxy properties. Furthermore, we have built EmpiriciSN to be easily updated with additional or different catalogs by the user; as long as the data files are formatted in a manner similar to our default data set, an EMPIRICIST object can fit a model to any number of supernova and host properties. This allows the user to make adjustments to the provided SDSS/SNLS data set (such as using fluxes instead of magnitudes, for their more Gaussian uncertainties), extend the model to include additional catalogs, or even to model different types of supernovae, in the future.

---

9 https://github.com/LSSTDESC/Twinkles
Testing using the BIC test described in Section 2.2.1 indicated that the preferred number of Gaussian components for our data set was six, and we have provided a default six-component model that is available with the EmpiriciSN software. The default model also has built-in labels for each of the SN and host parameters so that these can be used for conditioning, as described in Section 2.3.

3.2 Demo/Results

The EmpiriciSN class object is called EMPIRICIST, and here we demonstrate some of its basic functionality. If using the default model fit to our SDSS and SNLS data sample, or another model that has already been fit to a data set, it is straightforward to declare a new EMPIRICIST object and read in the existing model, as shown in Listing 6. A model can be loaded upon creation of an EMPIRICIST object by passing the model file name to the MODEL_FILE argument of the constructor. Alternatively, Listing 7 demonstrates how to run a test for the optimal number of components and fit a new model to a data set. The FIT_MODEL function always saves the newly fit model to a file, either with the default name “empiriciSN_model.fit” or with a file name passed into the function’s FILENAME argument. After fitting a model, that model is stored in the EMPIRICIST object and can be used for SN prediction without the need to load the model again.

Listing 6—An example of creating a new EMPIRICIST object and reading in our pre-computed default model. The EMPIRICIST object can also be created with the model already loaded by passing the file name to the MODEL_FILE argument of the constructor.

```python
from empiriciSN import Empiricist
emp = Empiricist()
emp.read_model('default_model.fit')
```

Listing 7—A demonstration of the interface for testing of the optimal number of components to use for a data set and fitting a model to that data set. The COMPONENT_TEST function’s third argument is the range of values for the number of Gaussian components to test using the BIC test built into the XDGM model; in this case we use the values 1–8. The newly fit model is automatically saved to a file with the default name “empiriciSN_model.fit,” or with a name passed into the FIT_MODEL function’s FILENAME argument.

```python
X, Xerr = (data, errors)
comp_range = np.array([1, 2, 3, 4, 5, 6, 7, 8])
bics, optimal_n_comp, lowest_bic =
emp.component_test(X=X, Xerr=Xerr,
                   comp_range=comp_range)
emp.fit_model(X=X, Xerr=Xerr,
              n_components=optimal_n_comp)
```

If using the default model, the local surface brightnesses at the location of the supernova in each filter are five of the host properties necessary for sampling a realistic SN for a host. However, the separation of the SN from the host nucleus is one of the properties being fit, and the local surface brightness cannot be calculated until a location has been chosen. Because of this, the prediction of a realistic SN proceeds in two steps: first, a position is selected based on a subset of host parameters, and then, the local surface brightness is calculated and the SALT2 parameters of the SN are sampled from a model conditioned on the full range of host parameters.

In order to select a location for an SN, the user can condition on any subset of the parameters used to fit the model. The indices of the parameters to condition on must be passed as an argument to the GET_LOGR function, as must the index of the log $R/R_e$ parameter. Any indices may be used except the first three, which are assumed to be the three SALT2 parameters for the SN. The EMPIRICIST object will condition the XDGM model using the data passed into the function and then return a value of log $R/R_e$ sampled from the conditioned model. Uncertainties on the quantities being used for conditioning the model may be used, but are not required. Listing 8 demonstrates this functionality, using the host galaxy redshift and 10 host colors to condition the model and select a location for the SN.

When using our default model, the GET_LOCAL_SB function can be used to select a local surface brightness at the location of the SN once a location has been sampled for a given host. The host data passed into this function must be an array of 21 surface brightness parameters. The first index should be the host Sérsic index (1 for an exponential profile, 4 for a de Vaucouleurs profile), and this should be followed by sets of magnitude, magnitude uncertainty, effective (half light) radius in arcseconds, and radius uncertainty for each of the five ugriz filters. The function returns two arrays, one with the local surface brightness in units of magnitudes per square arcsecond for each filter and the other containing the uncertainties on the local surface brightnesses. The magnitudes are assumed to be K-corrected and corrected for Galactic extinction, as this is what the default model was trained on. Listing 9 demonstrates this.

Listing 8—A demonstration of selecting a location for an SN given a subset of host parameters using our default model. In this case, the indices used for conditioning represent the redshift and the 10 ugriz colors of the host. The X and Xerr arrays represent arrays of the host redshift and colors and uncertainties on these quantities, respectively.

```python
X, Xerr = (data, errors)
cond_ind = np.array([3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14])
logR = emp.get_logR(cond_indices=cond_ind,
                 R_index=4, X=X, Xerr=Xerr)
```

Listing 9—A demonstration of calculating the local surface brightness of the host, if using our default model. The SB_PARAMS array is assumed to be of the form described in the text. The log R value is assumed to have been fit using GET_LOGR as demonstrated in Listing 8.

```python
SB, SB_err = emp.get_local_SB(SB_params=SB_params, R=logR)
```

Finally, once we have a location and local surface brightness, it is straightforward to sample an SN for the host galaxy using the GET_SN function. In Listing 10, we demonstrate how to select a single SN for a given host. The resulting array contains the SALT2 x0, x1, and c parameters for the sampled SN.

Figure 7 shows a plot of the measured SALT2 parameter distributions from our test SN sample in black, the SALT2 parameters for 482 SNe sampled from our unconditioned fitted
model in red, and the distribution of parameters for 1000 SNe
that were sampled for a single host galaxy in our sample in
blue, with location and local surface brightness calculated as
described above. As can be seen in the figure, the distribution
of supernova parameters sampled from the unconditioned
model matches that of the actual distribution quite well, given
the small numbers involved. Conversely, the distributions for
the various properties can be quite different for a single
position in a single host than what they are for the full sample,
which is the behavior we expect.

Listings 10—The supernova selection interface for the
EMPRIEST object. Here, we combine the radius and the host
properties fit in the previous two Listings with our initial
magnitude and redshift arrays to create an array with all of the
host properties needed to condition the model and sample
supernova properties. Here, we are using the array index
version of XDGM propagation conditioning as opposed to the dictionary
version.

```python
def host(params, host_params, X, logR, SB):
    host_params = np.array([np.nan, np.nan, np.nan])
    host_params = np.append(host_params, X[1:]
    host_params = np.append(host_params, SB)
    host_err = np.array([0.0, 0.0, 0.0])
    host_err = np.append(host_err, 0.0)
    host_err = np.append(host_err, Xerr[1:]
    host_err = np.append(host_err, SB_err)
    predicted_SN = emp.get_SN(X = host_params,
Xerr = host_err,
    n_SN = 1)
```

For a more detailed walkthrough of the EmpiriciSN software, please see the demo notebook on the EmpiriciSN github page.

4. Discussion

In this paper, we have summarized the capabilities of two
new pieces of Python software that implement Extreme
Deconvolution Gaussian Mixture Modeling for density estimation
in astrophysical contexts.

XDGM is a new implementation of XDGM that extends
existing XD algorithms and has several new capabilities that
are not present in any previously existing XDGM implementa-
tion. It allows the user to choose between either the
ASTROML (Vanderplas et al. 2012; Ivezic et al. 2015) or the
EXTREME-DECONVOLUTION code of Bovy et al. (2011) for
performing XDGM fits to data, and it uses the ASTROML
interface for fitting and sampling so that existing code using
ASTROML does not need to be modified. It extends the SCIKIT-
LEARN BASEESTIMATOR class so that cross-validation methods
will work, and it also has a function for computing the
Bayesian Information Criterion (BIC) of a model given a
certain data set, in order to allow the user to test different model
parameters. Finally, and most crucially, XDGM models can
be conditioned on a given subset of data, and the conditioned
model can then be used to sample the remaining parameters.
This allows our XDGM class to function as a prediction tool,
which will have a wide variety of astronomical applications.
XDGM is easily extendible to include additional implementa-
tions of XDGM methods or even different types of fitting
solutions altogether, such as a Dirichlet process GMM. As long
as the basic distribution remains a multivariate Gaussian, our
machine learning and conditioning algorithms should still
function as intended.

EmpiriciSN is an example use of XDGM as a prediction
and is designed to predict realistic supernova parameters
for a given set of host galaxy parameters. We have built an
XDGM model based on four SN Ia parameters (The SALT2
x0, x1, and color parameters and the location of the supernova
in the host galaxy) and 16 host galaxy parameters (redshift, ugriz
colors, sersic index, and ugriz local surface brightness at
the location of the SN) that is trained on a data set comprising
159 supernovae from SNLS and 1273 supernovae from SDSS.
Given the redshift, ugriz magnitudes, and surface brightness
profiles for a galaxy, EmpiriciSN can select a location of a
supernova relative to the effective radius of the supernova,
compute the local surface brightness in all five ugriz filters,
and sample realistic SALT2 parameters for the supernova given
the host properties using this default model. The user can also
train a new model based on a new data set, if the default data
set is not desired, or to change the parameters used in the
model. This makes it capable of performing similar predictions
for other types of supernovae, or performing simpler catalog
level simulations, leaving out the SN position and local surface
brightness information. EmpiriciSN was primarily built to be
used for planting realistic supernovae in simulated survey data
and is already being implemented for this purpose to model
LSST supernovae. In the near future, we also intend to combine
this tool with realistic large area galaxy mock catalogs, with
potential applications for various current and future surveys
including DES, DESI, and LSST.
Both XDGMM and EmpiriSN are open-source projects. We welcome further contributions and collaboration from the community.

We thank Rahul Biswas and Bob Nichol for useful discussion and feedback, and the three referees (domain reviewer, statistics editor, and data editor) for their positive and constructive feedback, which led to improvements to both the paper and the code. T.W.-S.H. was supported by the DOE Computational Science Graduate Fellowship, grant number DE-FG02-97ER25308. P.J.M. and R.H.W. acknowledge support from the U.S. Department of Energy under contract number DE-AC02-76SF00515.

Funding for SDSS-III has been provided by the Alfred P. Sloan Foundation, the Participating Institutions, the National Science Foundation, and the U.S. Department of Energy Office of Science. The SDSS-III web site is http://www.sdss3.org/.

References

Alam, S., Albareti, F. D., Allende Prieto, C., et al. 2015, ApJS, 219, 12
Astier, P., Guy, J., Regnault, N., et al. 2006, A&A, 447, 31
Barbary, K. 2014, sncosmo, v0.4.2, Zenodo, doi:10.5281/zenodo.11938
Betoule, M., Kessler, R., Guy, J., et al. 2014, A&A, 568, A22
Bishop, C. M. 2006, Pattern Recognition and Machine Learning (Heidelberg: Springer)
Blanton, M. R., & Roweis, S. 2007, AJ, 133, 734
Bovy, J., & Hogg, D. W. 2010, ApJ, 717, 617
Bovy, J., Hogg, D. W., & Roweis, S. T. 2009, ApJ, 700, 1794
Bovy, J., Hogg, D. W., & Roweis, S. T. 2011, AnApJS, 5, 1657
Bovy, J., Myers, A. D., Hennawi, J. F., et al. 2012, ApJ, 749, 41
Childress, M., Aldering, G., Antilogus, P., et al. 2013, ApJ, 770, 108
Connolly, A. J., Angeli, G. Z., Chandrasekharan, S., et al. 2014, Proc. SPIE, 9150, 915014
Galbany, L., Stanishev, V., Mourão, A. M., et al. 2014, A&A, 572, A38
Graur, O., Bianco, F. B., Huang, S., et al. 2016a, arXiv:1609.02921
Graur, O., Bianco, F. B., Modjaz, M., et al. 2016b, arXiv:1609.02923
Graur, O., Rodney, S. A., Maoz, D., et al. 2014, ApJ, 783, 28
Guy, J., Astier, P., Baumont, S., et al. 2007, A&A, 466, 11
Hogg, D. W., Blanton, M. R., Roweis, S. T., & Johnston, K. V. 2005, ApJ, 629, 268
Holoien, T. W.-S., Marhsall, P. J., & Wechsler, R. H. 2016a, XDGMM, v1.0, Zenodo, doi:10.5281/zenodo.163858
Holoien, T. W.-S., Marhsall, P. J., & Wechsler, R. H. 2016b, empiriciSN, v1.0, Zenodo, doi:10.5281/zenodo.163859
Ivezić, Z., Connolly, A. J., & Vanderplas, J. 2015, in AAS Meeting 225 Abstracts, #336.48
Ivezić, Ž, Connolly, A. J., VanderPlas, J. T., & Gray, A. 2014, Statistics Data Mining, and Machine Learning in Astronomy (Princeton, NJ: Princeton Univ. Press)
Ivezić, Ž, Smith, J. A., Miknaitis, G., et al. 2007, AJ, 134, 973
Ivezić, Z., Tyson, J. A., Abel, B., et al. 2008, arXiv:0805.2366
Koposov, S. E., Belokurov, V., & Wyn Evans, N. 2013, ApJ, 766, 79
Melinder, J., Mattila, S., Östlin, G., Mencía-Trinchant, L., & Fransson, C. 2008, A&A, 490, 419
Modjaz, M., Kewley, L., Kirshner, R. P., et al. 2008, AJ, 135, 1136
Pedregosa, F., Varoquaux, G., Gramfort, A., et al. 2011, Journal of Machine Learning Research, 12, 2825
Rasmussen, C. E., & Williams, C. K. I. 2006, Gaussian Processes for Machine Learning (Cambridge, MA: MIT Press)
Sako, M., Bassett, B., Becker, A. C., et al. 2014, arXiv:1401.3317
Schwarz, G. 1978, AnSta, 6, 461
Skuljan, J., Hearnshaw, J. B., & Cottrell, P. L. 1999, MNRAS, 308, 731
Sullivan, M., Conley, A., Howell, D. A., et al. 2010, MNRAS, 406, 782
Sullivan, M., Guy, J., Conley, A., et al. 2011, ApJ, 737, 102
Vanderplas, J., Connolly, A., Ivezić, Ž, & Gray, A. 2012, in Conf. Intelligent Data Understanding (CIDU), ed. K. Das, N. V. Chawla, & A. N. Srivastava (Boulder, CO: National Center for Atmospheric Research), 47
York, D. G., Adelman, J., Anderson, J. E., Jr., et al. 2000, AJ, 120, 1579