

## MACHINE LEARNING IN ASTRONOMY: A PRACTICAL OVERVIEW

DALYA BARON<sup>1</sup>

<sup>1</sup>*School of Physics and Astronomy  
Tel-Aviv University  
Tel Aviv 69978, Israel*

### ABSTRACT

Astronomy is experiencing a rapid growth in data size and complexity. This change fosters the development of data-driven science as a useful companion to the common model-driven data analysis paradigm, where astronomers develop automatic tools to mine datasets and extract novel information from them. In recent years, machine learning algorithms have become increasingly popular among astronomers, and are now used for a wide variety of tasks. In light of these developments, and the promise and challenges associated with them, the IAC Winter School 2018 focused on big data in Astronomy, with a particular emphasis on machine learning and deep learning techniques. This document summarizes the topics of supervised and unsupervised learning algorithms presented during the school, and provides practical information on the application of such tools to astronomical datasets. In this document I cover basic topics in supervised machine learning, including selection and preprocessing of the input dataset, evaluation methods, and three popular supervised learning algorithms, Support Vector Machines, Random Forests, and shallow Artificial Neural Networks. My main focus is on unsupervised machine learning algorithms, that are used to perform cluster analysis, dimensionality reduction, visualization, and outlier detection. Unsupervised learning algorithms are of particular importance to scientific research, since they can be used to extract new knowledge from existing datasets, and can facilitate new discoveries.

*Keywords:* methods: data analysis, methods: statistical

arXiv:1904.07248v1 [astro-ph.IM] 15 Apr 2019

## 1. CONTEXT

Astronomical datasets are undergoing a rapid growth in size and complexity, thus introducing Astronomy to the era of big data science (e.g., Ball & Brunner 2010; Pesenson et al. 2010). This growth is a result of past, ongoing, and future surveys, that produce massive multi-temporal and multi-wavelength datasets, with a wealth of information to be extracted and analyzed. Such surveys include the Sloan Digital Sky Survey (SDSS; York et al. 2000), which provided the community with multi-color images of  $\sim 1/3$  of sky, and high-resolution spectra of millions of Galactic and extra-galactic objects. Pan-STARRS (Kaiser et al. 2010) and the Zwicky Transient Facility (Bellm 2014) perform a systematic exploration of the variable sky, delivering time-series of numerous asteroids, variable stars, supernovae, active galactic nuclei, and more. Gaia (Gaia Collaboration et al. 2016) is charting the three-dimensional map of the Milky Way, and will provide accurate positional and radial velocity measurements for over a billion stars in our Galaxy and throughout the Local Group. Future surveys, e.g., DESI (Levi et al. 2013), SKA (Dewdney et al. 2009), and LSST (Ivezic et al. 2008), will increase the number of available objects and their measured properties by more than an order of magnitude.

In light of this accelerated growth, astronomers are developing automated tools to detect, characterize, and classify objects using the rich and complex datasets gathered with the different facilities. Machine learning algorithms have gained increasing popularity among astronomers, and are widely used for a variety of tasks.

Machine learning algorithms are generally divided into two groups. Supervised machine learning algorithms are used to learn a mapping from a set of features to a target variable, based on example input-output pairs provided by a human expert (see e.g., Connolly

et al. 1995; Collister & Lahav 2004; Re Fiorentin et al. 2007; Mahabal et al. 2008; Daniel et al. 2011; Laurino et al. 2011; Morales-Luis et al. 2011; Bloom et al. 2012; Brescia et al. 2012; Richards et al. 2012; Krone-Martins et al. 2014; Masci et al. 2014; Miller 2015; Wright et al. 2015; Djorgovski et al. 2016; D’Isanto et al. 2016; Lochner et al. 2016; Castro et al. 2018; Naul et al. 2018; D’Isanto & Polsterer 2018; D’Isanto et al. 2018; Krone-Martins et al. 2018; Zucker & Giryes 2018; Delli Veneri et al. 2019; Ishida et al. 2019; Mahabal et al. 2019; Norris et al. 2019; Reis et al. 2019). Unsupervised learning algorithms are used to learn complex relationships that exist in the dataset, without labels provided by an expert. These can roughly be divided into clustering, dimensionality reduction, and anomaly detection (e.g., Boroson & Green 1992; Protopapas et al. 2006; D’Abrusco et al. 2009; Vanderplas & Connolly 2009; Sánchez Almeida et al. 2010; Ascasibar & Sánchez Almeida 2011; D’Abrusco et al. 2012; Meusinger et al. 2012; Fustes et al. 2013; Krone-Martins & Moitinho 2014; Baron et al. 2015; Hocking et al. 2015; Gianniotis et al. 2016; Nun et al. 2016; Polsterer et al. 2016; Baron & Poznanski 2017; Reis et al. 2018a,b). The latter algorithms are arguably more important for scientific research, since they can be used to extract new knowledge from existing datasets, and can potentially facilitate new discoveries.

In view of the shift in data analysis paradigms and associated challenges, the IAC Winter School 2018 focused on big data in Astronomy. It included both lectures and hands-on tutorials, which are publicly available through their website<sup>1</sup>. The school covered the following topics: (1) general overview on the use of machine learning techniques in Astronomy: past, present and perspectives, (2) data challenges and solutions in forthcoming surveys, (3) su-

<sup>1</sup> <http://www.iac.es/winterschool/2018/>

ervised learning: classification and regression, (4) unsupervised learning and dimensionality reduction techniques, and (5) shallow and deep neural networks. In this document I summarize the topics of supervised and unsupervised learning algorithms, with special emphasis on unsupervised techniques. This document is not intended to provide a rigorous statistical background, but rather to present practical information on popular machine learning algorithms and their application to astronomical datasets. Supervised learning algorithms are discussed in section 2, with an emphasis on optimization (section 2.1), input datasets (section 2.2), and three popular algorithms: Support Vector Machine (section 2.3), Decision Trees and Random Forest (section 2.4), and shallow Artificial Neural Networks (section 2.5). Unsupervised learning algorithms are discussed in section 3, in particular distance assignment (section 3.1), clustering algorithms (section 3.2), dimensionality reduction algorithms (section 3.3), and anomaly detection algorithms (section 3.4).

## 2. SUPERVISED LEARNING

Supervised machine learning algorithms are used to learn a relationship between a set of measurements and a target variable using a set of provided examples. Once obtained, the relationship can be used to predict the target variable of previously-unseen data. The main difference between traditional model fitting techniques and supervised learning algorithms is that in traditional model fitting the model is predefined, while supervised learning algorithms construct the model according to the input dataset. Supervised learning algorithms can, by construction, describe very complex non-linear relations between the set of measurements and the target variable, and can therefore be superior to traditional algorithms that are based on fitting of predefined models.

In machine learning terminology, the dataset consists of objects, and each object has mea-

sured features and a target variable. In Astronomy, the objects are usually physical entities such as stars or galaxies, and their features are measured properties, such as spectra or light-curves, or various higher-level quantities derived from observations, such as a variability period or stellar mass. The type of target variable depends on the particular task. In a *classification* task, the target variables are discrete (often called labels), for example, classification of spectra into stars or quasars. In a *regression* task, the target variable is continuous, for example, redshift estimation using photometric measurements.

Supervised learning algorithms often have *model parameters* that are estimated from the data. These parameters are part of the model that is learned from the data, are often saved as part of the learned model, and are required by the model when making predictions. Examples of model parameters include: support vectors in Support Vector Machines, splitting features and thresholds in Decision Trees and Random Forest, and weights of Artificial Neural Networks. In addition, supervised learning algorithms often have *model hyper-parameters*, which are external to the model and whose values are often set using different heuristics. Examples of model hyper-parameters include: the kernel shape in Support Vector Machines, the number of trees in Random Forests, and the number of hidden layers in Artificial Neural Networks.

The application of supervised learning algorithms is usually divided into three stages. In the *training stage*, the model hyper-parameters are set, and the model and the model parameters are learned from a subset of the input dataset, called the *training set*. In the *validation stage*, the model hyper-parameters are optimized according to some predefined cost function, often using a different subset of the input dataset, called the *validation set*. During

the validation stage, the model training is carried out iteratively for many different choices of hyper-parameters, and the hyper-parameters that result in the best performance on the validation set are chosen. Finally, in the *test stage*, the trained model is used to predict the target variable of a different subset of the input dataset, called the *test set*. The latter stage is necessary in order to assess the performance of the trained model on a previously-unseen dataset, i.e., a subset of the input data that was not used during the training and validation stages, and can be used to compare different supervised learning algorithms. Once these stages are completed, the model can be used to predict the target variable of new, previously-unseen datasets.

This section provides some basic principles of supervised machine learning, and presents several popular algorithms used in Astronomy. For a detailed review on the subject, see Biehl (2019). The section starts by describing the cost functions that are usually used to optimize model hyper-parameters, assess the performance of the final model, and compare different supervised learning algorithms (section 2.1). Then, section 2.2 gives additional details on the input dataset, in particular its partition to training, validation, and test sets, feature selection and normalization, and imbalanced datasets. Finally, three popular algorithms are presented: Support Vector Machine (section 2.3), Decision Trees and Random Forest (section 2.4), and shallow Artificial Neural Networks (section 2.5).

### 2.1. Evaluation Metrics

There are different evaluation metrics one can use to optimize supervised learning algorithms. Evaluation metrics are used to optimize the model hyper-parameters, to assess the performance of the final model, to select optimal subset of features, and to compare between different supervised learning algorithms. The evalua-

True label \ Predicted label	1	2	4	5	6	8	13
1	94	2	0	2	0	0	0
2	18	81	0	0	0	0	0
4	32	0	53	14	0	0	0
5	32	0	1	65	0	0	0
6	26	0	5	66	0	1	0
8	78	0	0	4	0	13	0
13	1	1	5	1	2	3	83

**Figure 1.** Example of a confusion matrix taken from Mahabal et al. (2017), who trained a deep learning model to distinguish between 7 classes of variable stars, marked by 1, 2, 4, 5, 6, 8, and 13 in the diagram. The confusion matrix shows the number of objects in each class versus the number of objects predicted by the model to belong to a particular class. In the best-case scenario, the confusion matrix will contain non-zero elements only in its diagonal, and zero elements otherwise.

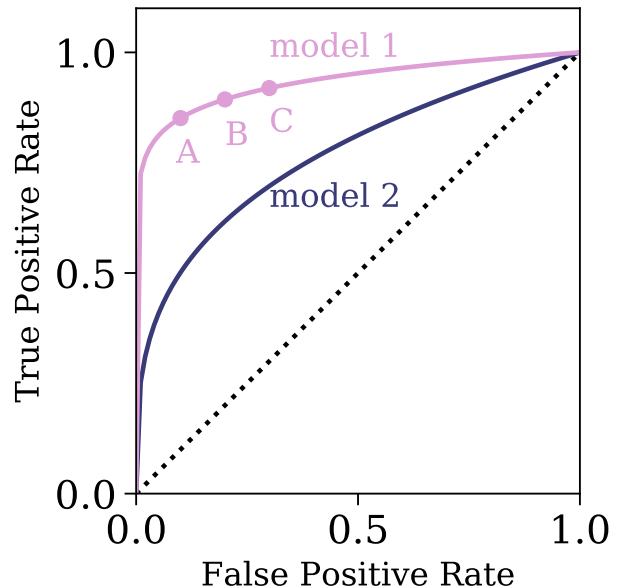
tion metrics are computed during the validation and the test stages, where the trained model is applied to a previously-unseen subset of the input dataset, and the target variable predicted by the model is compared to the target variable provided in the input data.

In regression tasks, where the target variable is continuous, the common metrics for evaluating the predictions of the model are the *Mean Absolute Error* (MAE) and the *Mean Squared Error* (MSE). The MAE is equal to  $\frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$ , where  $n$  is the number of objects in the validation or test set,  $\hat{y}_i$  is the target variable predicted by the model, and  $y_i$  is the target variable provided in the input data. The MSE is equal to  $\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$ . The

MSE has the disadvantage of heavily weighting outliers, since it squares each term in the sum, giving outliers a larger weight. When outlier weighting is undesirable, it is better to use the MAE instead. Finally, it is worth noting additional metrics used in the literature, for example, the *Normalized Median Absolute Deviation*, the *Continuous Rank Probability Score*, and the *Probability Integral Transform* (see e.g., D’Isanto & Polsterer 2018; D’Isanto et al. 2018).

In classification tasks, where the target variable is discrete, the common evaluation metrics are the *Classification Accuracy*, the *Confusion Matrix*, and the *Area Under ROC Curve*. Classification accuracy is the ratio of the number of correct predictions (i.e., the class predicted by the model is similar to the class provided in the input dataset) to the total number of predictions made. This value is obviously bound between 0 and 1. The accuracy should be used when the number of objects in each class is roughly similar, and when all predictions and prediction errors are equally important. Confusion matrices are used in classification tasks with more than two classes. Figure 1 shows an example of a confusion matrix, taken from Mahabal et al. (2017), who trained a supervised learning model to distinguish between 7 classes of variable stars. Each row and column in the matrix represents a particular class of objects, and the matrix shows the number of objects in each class versus the number of objects predicted by the model to belong to a particular class. In the best-case scenario, we expect the confusion matrix to be purely diagonal, with non-zero elements on the diagonal, and zero elements otherwise. Furthermore, similarly to the accuracy, one can normalize the confusion matrix to show values that range from 0 to 1, thus removing the dependence on the initial number of objects in each class.

Finally, the receiver operating characteristic curve (ROC curve) is a useful visualization tool



**Figure 2.** An illustration of an ROC curve, where the true positive rate is plotted against the false positive rate. In the best-case scenario, we expect the true positive rate to be 1, and the false positive rate to be 0. The black line represents the resulting ROC curve for random assignment of classes, which is the worst-case scenario. The diagram is populated by varying the model hyper-parameters and plotting the true positive rate versus the false positive rate obtained for the validation set. The pink curve represents the ROC curve of model 1, where A, B, and C represent three particular choices of hyper-parameter value. The purple curve represents the ROC curve of model 2. The area under the ROC curve can be used to select the optimal model, which, in this case, is model 1.

of a supervised algorithm performance in a binary classification task. Figure 2 shows an illustration of an ROC curve, where the *True Positive Rate* is plotted against the *False Positive Rate*. The true positive rate represents the number of “true” events that are correctly identified by the algorithm, divided by the total number of “true” events in the input dataset. The false positive rate represents the number of “false” events that were wrongly classified as “true” events divided by the total number of

”false” events. For example, if we are interested in detecting gravitational lenses in galaxy images, ”true” events are images with gravitational lenses, and ”false” events are images without gravitational lenses. In the best-case scenario, we expect the true positive rate to be 1, and the false positive rate to be 0. The ROC curve diagram is generated by varying the model hyper-parameters, and plotting the true positive rate versus the false positive rate obtained for the validation set (the continuous ROC curves presented in Figure 2 are produced by varying the classification/detection threshold of a particular algorithm. This threshold can be considered as a model hyper-parameter). Furthermore, the diagram can be used to compare the performance of different supervised learning algorithms, by selecting the algorithm with the maximal area under the curve. For example, in Figure 2, model 1 outperforms model 2 for any choice of hyper-parameters. Finally, depending on the task, one can decide to optimize differently. For some tasks one cannot tolerate false negatives (e.g., scanning for explosives in luggage), while for others the total number of errors is more important.

## 2.2. Input Dataset

The input to any supervised learning algorithm consists of a set of objects with measured features, and a target variable which can be either continuous or discrete. As noted in the introduction to this section, the input dataset is divided into three sets, the training, validation, and test sets. The model is initially fit to the training set. Then, the model is applied to the validation set. The validation set provides an unbiased evaluation of the model performance while tuning the model’s hyper-parameters. Validation sets are also used for regularization and to avoid overfitting, with the common practice of stopping the training process when the error on the validation dataset increases. Finally, the test set is used to pro-

vide an unbiased evaluation of the final model, and can be used to compare between different supervised learning algorithms. To perform a truly unbiased evaluation of the model performance, the training, validation, and test sets should be mutually exclusive.

The dataset splitting ratios depend on the dataset size and on the algorithm one trains. Some algorithms require a substantial amount of data to train on, forcing one to enlarge the training set at the expense of the others. Algorithms with a few hyper-parameters, which are easily validated and tuned, require small validation sets, whereas models with many hyper-parameters might require larger validation sets. In addition, in *Cross Validation* the dataset can be repeatedly split into training and validation sets, for example, by randomly selecting objects from a predefined set, and the model is then iteratively trained and validated on these different sets (see e.g., Miller et al. 2017). There are different splitting methods that are implemented in PYTHON and are publicly available in the SCIKIT-LEARN library<sup>2</sup>.

The performance of all supervised learning algorithms strongly depends on the input dataset, and in particular on the features that are selected to form the dataset. Most supervised learning algorithms are not constructed to work with hundreds or thousands of features, making feature selection a key part of the applied machine learning process. Feature selection can be done manually by an expert in the field, by defining features that are most probably relevant for the task at hand. There are various alternative ways to select an optimal set of features without domain knowledge, including filter methods, wrapper methods, and embedded methods. Filter methods assign a statistical score to each feature, and features are selected

<sup>2</sup> [https://scikit-learn.org/stable/model\\_selection.html](https://scikit-learn.org/stable/model_selection.html)

or removed according to this score. In wrapper methods, different combinations of features are prepared, and the combination that results in the best accuracy is selected. Embedded methods learn which features best contribute to the accuracy of the model during the model construction (see also Donalek et al. 2013; D’Isanto et al. 2016, 2018). Some of these methods are included in the SCIKIT-LEARN library<sup>3</sup>. Finally, a pertinent note on deep learning, in particular *Convolutional Neural Networks*. The structure of these networks allows them to take raw data as an input (e.g., spectra, light-curves, and images), and perform efficient feature extraction during the training process. Thus, using such models, there is usually no need to perform feature selection prior to the training stage.

Feature scaling is an additional key part of the data preparation process. While some algorithms do not require any feature scaling prior to training (e.g., Decision Trees and Random Forest), the performance of other algorithms strongly depends on it, and it is advised to apply some feature scaling prior to their training (e.g., for Support Vector Machine). There are various ways to scale the features in the initial dataset, including standardization, mean normalization, min-max scaling, and application of dimensionality reduction algorithms to the initial dataset. These will not be further discussed in this document, however, many feature scaling methods are available in SCIKIT-LEARN<sup>4</sup>.

Finally, it is worth noting the topic of imbalanced datasets. Imbalanced data typically refers to the problem of a classification task where the classes are not equally represented. During training, many supervised learning algorithms assign equal weights to all the objects in the sample, resulting in a good performance

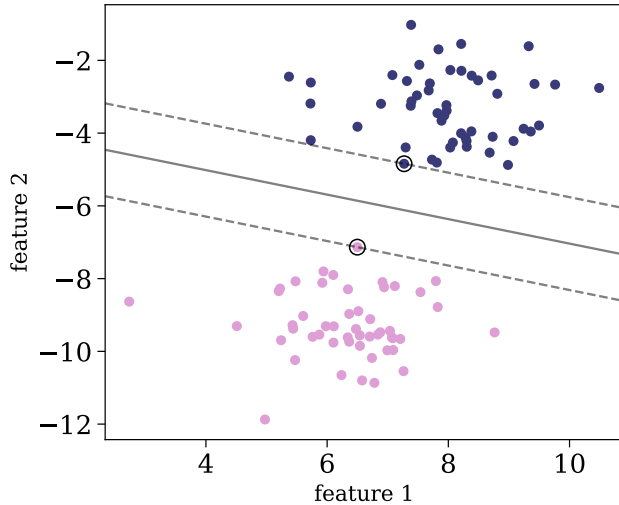
on the larger class, and worse performance on the smaller class. In addition, the regular *accuracy* cannot be used to evaluate the resulting model. Assume for example that we are interested in detecting gravitational lenses, and our dataset contains 100 000 images of galaxies, out of which 100 images show evidence of lensing. For such a dataset, an algorithm that classifies *all* objects as "not lens", regardless of the input features, will have an accuracy of 0.999. There are several methods to train and evaluate supervised learning algorithms in the presence of imbalanced datasets. One could apply different weights to different classes of objects during the training process, or undersample the larger class, or oversample the smaller class of objects. These, and additional methods, are available in SCIKIT-LEARN. Instead of the classification accuracy, one can use the ROC curve (figure 2), and select the hyper-parameters that result in the desired true positive versus false negative rates.

### 2.3. Support Vector Machine

One of the most popular supervised learning algorithms is Support Vector Machine (SVM), which has been applied in Astronomy for a variety of tasks (e.g., Qu et al. 2003; Huertas-Company et al. 2008; Fadely et al. 2012; Małek et al. 2013; Kovács & Szapudi 2015; Krakowski et al. 2016; Hartley et al. 2017; Hui et al. 2018; Ksoll et al. 2018; Pashchenko et al. 2018). Given a dataset with  $N$  features, SVM finds a hyperplane in the  $N$ -dimensional space that best separates the given classes. In a two-dimensional space, this hyperplane is a line that divides the plane into two parts, where every class lies on a different side. The optimal hyperplane is defined to be the plane that has the maximal margin, i.e the maximum distance between the plane and the data points. The latter are called the support vectors. Once obtained, the hyperplane serves as a decision boundary, and new objects are classified according to their location

<sup>3</sup> [https://scikit-learn.org/stable/modules/feature\\_selection.html](https://scikit-learn.org/stable/modules/feature_selection.html)

<sup>4</sup> <https://scikit-learn.org/stable/modules/preprocessing.html>



**Figure 3.** Illustration of the SVM best hyperplane for a two-dimensional dataset with linearly-separable classes. The two classes are plotted with pink and purple circles, and the support vectors are marked with black circles. The hyperplane is marked with a solid grey line.

with respect to the hyperplane. Figure 3 shows an illustration of the SVM hyperplane for a two-dimensional dataset, in which the classes are linearly-separable.

More often than not, the different classes in the dataset are not linearly-separable. The left panel of figure 4 shows an example of a two-dimensional dataset with two classes, which are not linearly-separable (i.e., the classes cannot be separated using a single straight line). The classification problem can be approached with the SVM *kernel trick*. Instead of constructing the decision boundary in the input data space, the dataset is mapped into a transformed feature space, which is of higher dimension, where linear separation might be possible. Once the decision boundary is found, it is back-projected to the original input space, resulting in a non-linear boundary. The middle panel of figure 4 shows the three-dimensional feature space that resulted from such a mapping, where, in this representation, the classes are linearly-separable. The right panel of figure 4 shows

the result of the back-projection of the decision boundary. To apply the kernel trick, one must define the *kernel function* that is related to the non-linear feature mapping. There is a wide variety of kernel functions, the most popular being Gaussian Radial Basis Function (RBF), Polynomial, and Sigmoid. The kernel function is a hyper-parameter of SVM, and these functions usually depend on additional parameters, which are also hyper-parameters of the model. SVM is available in `SCIKIT-LEARN`<sup>5</sup>

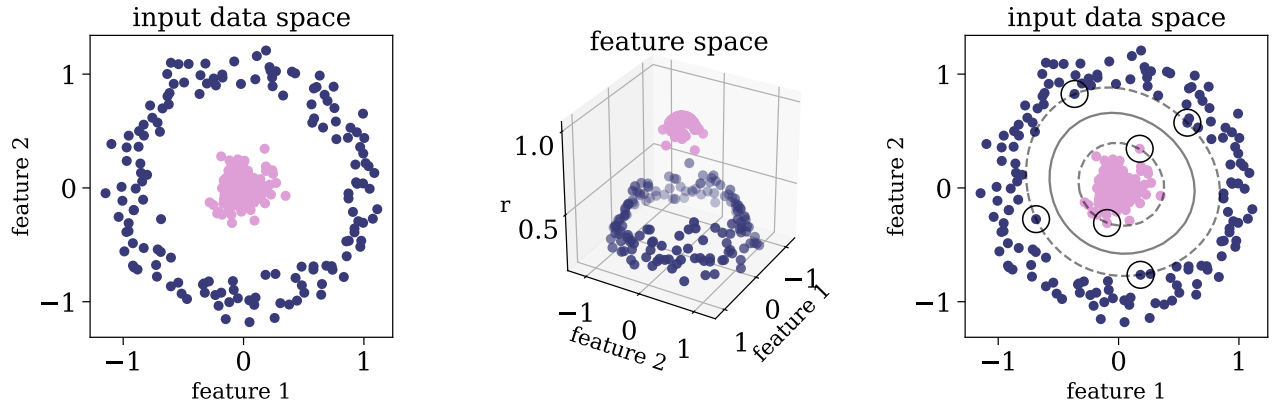
SVM is simple and robust, allowing one to classify a large variety of datasets and to construct very non-linear decision boundaries. Since SVM is based on measuring Euclidean distances between the objects in the sample and the hyperplane, it is very sensitive to feature scaling. Therefore, it is advised to scale the features. SVM can be applied to datasets with many features, but its performance might be strongly affected by the presence of irrelevant features in the dataset. It is therefore recommended to perform feature selection prior to the training.

#### 2.4. Decision Trees and Random Forest

Ensemble methods are meta-algorithms that combine several supervised learning techniques into a single predictive model, resulting in an overall improved performance, compared to the performance of each individual supervised algorithm. Ensemble methods either combine different supervised learning algorithms, or combine the information of a single algorithm that was trained on different subsets of the training set. One of the most popular ensemble methods is Random Forest, which is a collection of Decision Trees (Breiman et al. 1984; Breiman 2001). Random Forest is mainly used as a supervised algorithm for classification and regression (e.g., Carliles et al. 2010; Bloom et al. 2012; Pichara

<sup>5</sup> <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>





**Figure 4.** Application of the SVM *kernel trick* to a two-dimensional dataset that consists of two classes which are not linearly-separable. The left panel shows the dataset, where the different classes are represented by pink and purple circles. The middle panel shows the three-dimensional feature space that resulted from the applied mapping, where the classes can be separated with a two-dimensional hyperplane. The right panel shows the result of back-projecting the decision boundary to the input space, where the support vectors are marked with black circles, and the decision boundary with a solid grey line.

et al. 2012; Pichara & Protopapas 2013; Möller et al. 2016; Miller et al. 2017; Plewa 2018; Yong et al. 2018; Ishida et al. 2019), but can also be used in an unsupervised setting, to produce similarity measures between the objects in the sample (Shi & Horvath 2006; Baron & Poznanski 2017; Reis et al. 2018a,b).

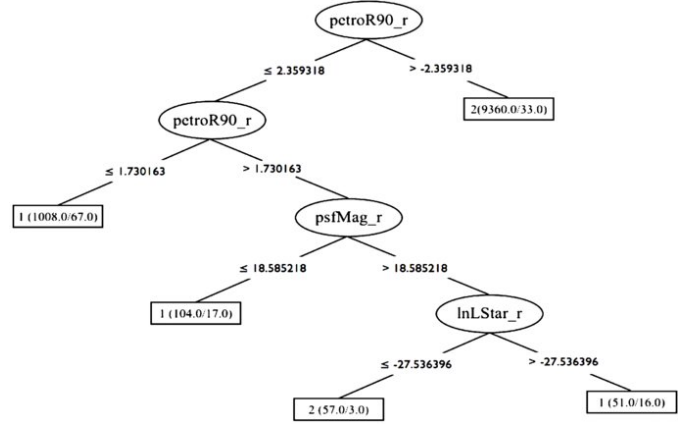
A decision tree is a non-parametric model constructed during the training stage, which is described by a top-to-bottom tree-like graph, and is used in both classification and regression tasks. The decision tree is a set of consecutive nodes, where each node represents a condition on one feature in the dataset. The conditions are of the form  $X_j > X_{j,th}$ , where  $X_j$  is the value of the feature at index  $j$  and  $X_{j,th}$  is some threshold, both of which are determined during the training stage. The lowest nodes in the tree are called terminal nodes or leaves, and they do not represent a condition, but instead carry the assigned label of a particular path within the tree. Figure 5 shows a visualization of a trained decision tree, taken from Vasconcellos et al. (2011), who trained decision tree classifiers to distinguish stars from galaxies.

To describe the training process of the decision tree, I consider the simple case of a classification task with two classes. The training process starts with the entire training set in the highest node of the tree – the root. The algorithm searches for the feature  $X_j$  and the feature threshold  $X_{j,th}$  that result in the best separation of the two classes, where the definition of best separation is a model hyper-parameter, with the typical choices being the *Gini impurity* or the *information gain*. Once the best feature and best threshold are determined, the training set propagates to the left and right nodes below the root, according to the established condition. This process is repeated recursively, such that deeper nodes split generally smaller subsets of the original data. In its simplest version, the recursive process stops when every leaf of the tree contains a single class of objects. Once the decision tree is trained, it can be used to predict the class of previously-unseen objects. The prediction is done by propagating the object through the tree, according to its measured features and the conditions in the nodes. The predicted class of the object is then the label of the terminal leaf (for additional information see

Breiman et al. 1984; Vasconcellos et al. 2011; Reis et al. 2019).

Decision trees have several advantages. First, in their simpler forms, they have very few hyper-parameters, and can be applied to large datasets with numerous features. Second, their recursive structures are easily interpretable, in particular, they can be used to determine the feature importance. Feature importance represents the relative importance of different features to the classification task at hand. Since the trees are constructed recursively with the aim of splitting the dataset into the predefined classes, features that are selected earlier in the training process, closer to the root, are more important than features that are selected later, closer to the terminal nodes. Obviously, features that were not selected in any node during the training process, carry little relevant information to the classification task. In more complex versions, decision trees can provide a measure of uncertainty for the predicted classes (see e.g., Breiman et al. 1984; Vasconcellos et al. 2011). However, in the simplest version, there are no restrictions on the number of nodes or the depth of the resulting tree, making the algorithm extremely sensitive to outliers. The resulting classifier will typically show a perfect performance on the training set, but a poor performance on new previously-unseen datasets. A single decision tree is typically prone to overfitting the training data, and cannot generalize to new datasets. Therefore, it is rarely used in its single form.

Random Forest is a collection of decision trees, where different decision trees are trained on different randomly-selected subsets of the original training set, and during the training of each individual tree, random subsets of the features are used to construct the conditions in individual nodes. This randomness reduces the correlation between the different trees, resulting in somewhat different tree structures with different conditions in their nodes. The Random Forest pre-



**Figure 5.** An example of a trained decision tree, taken from Vasconcellos et al. (2011). The decision tree contains nodes which represent conditions on features from the dataset, in this case `petroR90_r`, `psfMag_r`, and `nLStar_r`. The terminal nodes represent the assigned label of each particular path within the tree, which are 1, 1, 2, 1, and 2 from left to right, and represent stars and galaxies.

diction is an aggregate of individual predictions of the trees in the forest, in the form of a majority vote. That is, a previously-unseen object is propagated through the different trees, and its assigned label is the label reached in the majority of the trees. While a single decision tree tends to overfit the training data, the combination of many decision trees in the form of a Random Forest generalizes well to previously unseen datasets, resulting in a better performance (for additional information see Breiman 2001). As previously noted, Random Forest is one of the most popular machine learning algorithms in Astronomy.

Random Forest can be applied to datasets with thousands of features, with a moderate increase in running time. The algorithm has a handful of hyper-parameters, such as the number of trees in the forest and the number of randomly-selected features to consider in each node in each tree. In terms of performance, model complexity, and number of hyper-parameters, Random Forest is between SVM and Deep Neural Networks. The main disad-

vantage of Random Forest, which applies to most supervised learning algorithms, is its inability to take into account feature and label uncertainties. This topic is of particular interest to astronomers, and I discuss it below in section 2.4.1.

Finally, it is worth noting that ensemble methods rely on the construction of a *diverse* collection of classifiers and aggregation of their predictions (see e.g., [Kuncheva & Whitaker 2003](#)). Ensemble methods in the form of "bagging" tend to decrease the classification variance, while methods in the form of "boosting" tend to decrease the classification bias. Random Forest is a "bagging" ensemble method, where the ensemble consists of a diverse collection of individual trees that are trained on different subsets of the data. In "boosting", the decision trees are built sequentially, such that each tree is presented with training samples that the previous tree failed to classify. One of the most popular "boosting" methods in Astronomy is Adaboost ([Freund & Schapire 1997](#)). The three algorithms described in this section are available in the SCIKIT-LEARN library<sup>6</sup>.

#### 2.4.1. Probabilistic Random Forest

While shown to be very useful for various tasks in Astronomy, many Machine Learning algorithms were not designed for astronomical datasets, which are noisy and have gaps. In particular, measured features typically have a wide range of uncertainty values, and these uncertainties are often not taken into account when training the model. Indeed, the performance of

Machine Learning algorithms depends strongly on the signal-to-noise ratio of the objects in the sample, and a model optimized on a dataset with particular noise characteristics will fail on a similar dataset with different noise properties. Furthermore, while in computer vision the labels provided to the algorithm are considered to be "ground truth" (e.g., classification of cats and dogs in images), in Astronomy the labels might suffer from some level of ambiguity. For example, in a classification task of "real" versus "bogus" in transient detection on difference images (see e.g., [Bloom et al. 2012](#)), the labels in the training set are obtained from a manual classification of scientists and citizen-scientists. While some might classify a given event as "real", others may classify it as "bogus". In addition, labels in the training set could be the output of a different algorithm, which provides a label with an associated uncertainty. Such uncertainties are also not treated by most Machine Learning algorithms.

Recently, [Reis et al. \(2019\)](#) modified the traditional Random Forest algorithm to take into account uncertainties in the measurements (i.e., features) as well as in the assigned class. The *Probabilistic Random Forest* algorithm treats the features and the labels as random variables rather than deterministic quantities, where each random variable is represented by a probability distribution function, whose mean is the provided measurement and its variance is the provided uncertainty. Their tests showed that the Probabilistic Random Forest outperforms the traditional Random Forest when applied to datasets with various noise properties, with an improvement of up to 10% in classification accuracy with noisy features, and up to 30% with noisy labels. In addition to the dramatic improvement in classification accuracy, the Probabilistic Random Forest naturally copes with missing values in the data, and outperforms Random Forest when applied to a dataset with

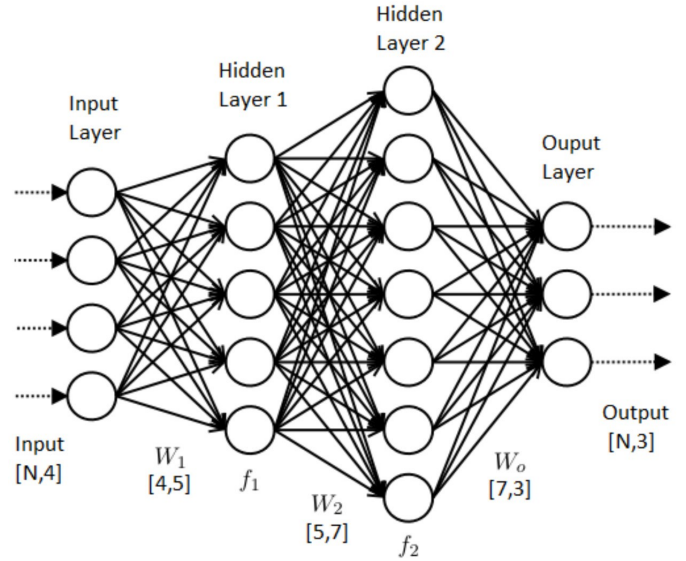
<sup>6</sup> <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>  
<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>  
<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html>

different noise characteristics in the training and test sets. The Probabilistic Random Forest was implemented in PYTHON and is publicly available on GITHUB<sup>7</sup>.

### 2.5. Artificial Neural Networks

Artificial neural networks are a set of algorithms with structures that are vaguely inspired by the biological neural networks that constitute the human brain. Their flexible structure and non-linearity allows one to perform a wide variety of tasks, including classification and regression, clustering, and dimensionality reduction, making them extremely popular in Astronomy (e.g., Storrie-Lombardi et al. 1992; Weaver & Torres-Dodgen 1995; Singh et al. 1998; Snider et al. 2001; Firth et al. 2003; Tagliaferri et al. 2003; Vanzella et al. 2004; Blake et al. 2007; Banerji et al. 2010; Eatough et al. 2010; Brescia et al. 2013, 2014; Ellison et al. 2016; Teimoorinia et al. 2016; Mahabal et al. 2017; Bilicki et al. 2018; Huertas-Company et al. 2018; Naul et al. 2018; Parks et al. 2018; Das & Sanders 2019). In this section I describe the main building blocks of *shallow* artificial neural networks, and their use for classification and regression tasks. The section does not include details on *Deep Learning*, in particular *Convolutional Neural Networks*, *Recurrent Neural Networks*, and *Generative Adversarial Networks* (see lectures by M. Huertas-Company for details on deep learning).

Figure 6 is an illustration of a shallow neural network architecture. The network consists of an *input layer*, *output layer*, and several *hidden layers*, where each of these contain neurons that transmit information to the neurons in the succeeding layer. The input data is transmitted from the input layer, through the hidden layers, and reaches the output layer, where the target variable is predicted. The value of every neuron in the network (apart from the neurons



**Figure 6.** Illustration of a shallow neural network architecture. The network consists of an input layer, two hidden layers, and an output layer. The input dataset is propagated from the input layer, through the hidden layers, to the output layer, where a prediction of a target variable is made. Each neuron is a linear combination of the neuron values in the previous layer, followed by an application of a non-linear activation function (see text for more details).

in the input layer) is a linear combination of the neurons in the previous layer, followed by an application of a non-linear activation function. That is, the values of the neurons in the first hidden layer are given by  $\vec{x}_1 = f_1(W_1\vec{x}_0)$ , where  $\vec{x}_0$  is a vector that describes the values of the neurons in the input layer (the input data),  $W_1$  is a weight matrix that describes the linear combination of the input values, and  $f_1$  is a non-linear activation function. In a similar manner, the values of the neurons in the second hidden layer are given by  $\vec{x}_2 = f_2(W_2\vec{x}_1)$ , with a similar notation. Finally, the values of the neurons in the output layer are given by  $\vec{x}_3 = f_3(W_3\vec{x}_2) = f_3(W_3f_2(W_2f_1(W_1\vec{x}_0)))$ . The weights of the network are model parameters which are optimized during training via *back-propagation* (for additional details see lectures by M. Huertas-Company). The non-linear ac-

<sup>7</sup> <https://github.com/ireis/PRF>

tivation functions are model hyper-parameters, with common choices being sigmoid, rectified linear unit function (RELU), hyperbolic tan function (TANH), and softmax. The number of hidden layers and the number of neurons in each of these layers are additional hyper-parameters of the model. The number of neurons in the input and the output layers are defined according to the classification or regression task at hand. Shallow neural networks are available in SCIKIT-LEARN<sup>8</sup>.

Owing to their flexible structure and non-linearity, artificial neural networks are powerful algorithms, capable of describing extremely complex relations between the input data and the target variable. As just noted, these networks have many hyper-parameters, and they usually require a large amount of data to train on. Furthermore, due to their complexity, they tend to overfit the dataset, and various techniques, such as *dropout*, are applied to overcome this issue. In addition, these networks are harder to interpret, compared to SVM or Random Forest. However, studies have shown that deep neural networks can greatly outperform traditional algorithms such as SVM, Random Forest, and shallow neural networks, given raw and complex data, such as images, spectra, and light-curves (see e.g., Huertas-Company et al. 2018; Naul et al. 2018; Parks et al. 2018 and references within).

### 3. UNSUPERVISED LEARNING

Unsupervised Learning is a general term that incorporates a large set of statistical tools, used to perform data exploration, such as clustering analysis, dimensionality reduction, visualization, and outlier detection. Such tools are particularly important in scientific research, since they can be used to make new discoveries or extract new knowledge from the dataset. For

example, a cluster analysis that reveals two distinct clusters of planets might suggest that the two populations are formed through different formation channels, or, a successful embedding of a complex high-dimensional dataset onto two dimensions might suggest that the observed complexity can be attributed to a small number of physical parameters (e.g., the large variety of stellar spectra can be attributed to a single sequence in temperature, stellar mass, and luminosity; e.g., Hertzsprung 1909 and Russell 1914). While visual inspection of the dataset can achieve these goals, it is usually limited to 3–12 dimensions (see lecture 2 by S. G. Djorgovski). Visual inspection becomes impractical with modern astronomical surveys, which provide hundreds to thousands of features per object. It is therefore necessary to use statistical tools for this task.

Unsupervised Learning algorithms take as an input only the measured features, without labels, and as such, they cannot be trained with some "ground truth". Their output is typically a non-linear and non-invertible transformation of the input dataset, consisting of an association of different objects to different clusters, low-dimensional representation of the objects in the sample, or a list of peculiar objects. Such algorithms consist of internal choices and a cost function, which does not necessarily coincide with our scientific motivation. For example, although K-means algorithm is often used to perform clustering analysis (see section 3.2.1), it is not optimized to detect clusters, and it will reach a correct global optimum even if the dataset is not composed of clusters. In addition, these algorithms often have several *external* free parameters, which cannot be optimized through the algorithm's cost function. Different choices of external parameters may result in significantly different outputs, resulting in different interpretations of the same dataset. The interpretation of the output of an unsu-

<sup>8</sup> [https://scikit-learn.org/stable/modules/neural\\_networks\\_supervised.html](https://scikit-learn.org/stable/modules/neural_networks_supervised.html)

ervised learning algorithm must be carried out with extreme caution, taking into account its internal choices and cost function, and the output’s sensitivity to a change of external parameters. However, since unsupervised learning is typically used for exploration, not necessarily to reach a precisely determined goal, the lack of objective to optimize is often not critical. It does make it hard to know when one has really exhausted exploring all possibilities.

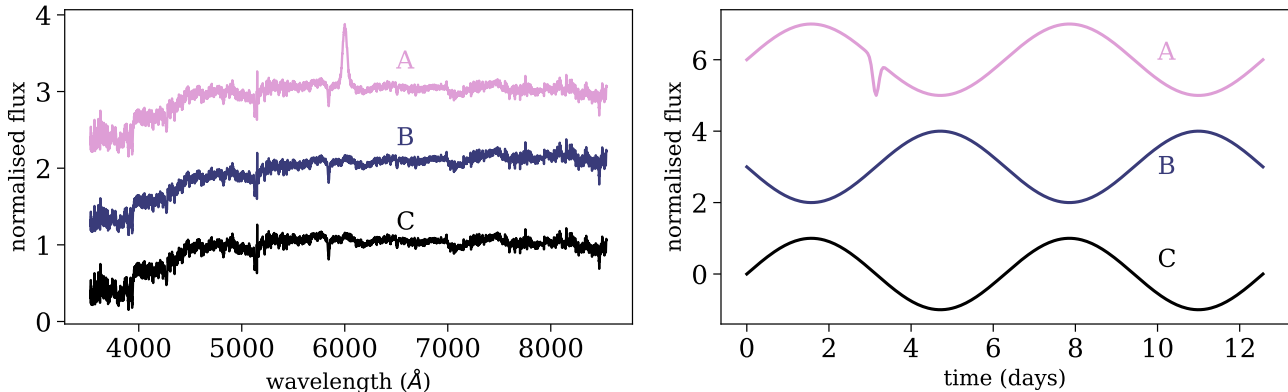
### 3.1. Distance Assignment

The first step in the large majority of unsupervised learning algorithms is distance assignment between the objects in the sample. In most of the cases it is assumed that the measured features occupy a euclidean space, and the distance between the objects in the sample is measured with euclidean metric. This choice might not be appropriate for some astronomical datasets, and using a metric that is more appropriate could improve the algorithm’s performance. For example, when the input dataset consists of features that are extracted from astronomical observations, these features typically do not have the same physical units (e.g., stellar mass, temperature, size and morphology of a galaxy, bolometric luminosity, etc). In such cases, euclidean distance assignment will be dominated by features that are distributed over the largest dynamical range (e.g., black hole mass  $\sim 10^8 M_\odot$ ), and will not be sensitive to the other features in the input dataset (e.g., stellar velocity dispersion  $\sim 200$  km/sec). Therefore, when applying unsupervised machine learning algorithms to a set of derived features, it is advised to rescale the features (see also section 2.2).

When applied to astronomical observations, such as spectra, light-curves, or images, the euclidean distance might not trace the distances a scientist would assign. This is partly since the euclidean metric implicitly assumes that all the features are equally important, which is not necessarily the case. Figure 7 shows two examples

of such scenarios. The left panel shows spectra of three galaxies, marked by A, B, and C, and the right panel shows three synthetic light-curves. In the galaxy case, galaxy A appears different, since it shows a strong emission line. Thus, we would expect that the distance between galaxy A and B will be larger than the distance between galaxy B and C. However, most of the pixels in a galaxy spectrum are continuum pixels, and the continuum of galaxy B is slightly bluer than the continuum of galaxy C. Since the spectra are dominated by continuum pixels, the euclidean distance between galaxy B and C will be *larger* than the distance to galaxy A. That is, using a euclidean metric, galaxy B is different. Similarly in the light-curve case, since the euclidean metric is sensitive to horizontal shifts, the distance between light-curve B and C will be larger than the distance between light-curve A and C. In many scientific applications, we want to define a metric in which A stands out. Such metrics can be based on domain knowledge, where e.g., the metric is invariant to shifts, rotations, and flips, or they can be based on some measure of feature importance, e.g., emission line pixels being more important than continuum pixels.

There are several distance measures which can be more appropriate for astronomical datasets and may result in improved performance, such as cross correlation-based distance (see e.g., Protopapas et al. 2006; Nun et al. 2016; Reis et al. 2018b), Distance Correlation, Dynamic Time Warp, Canberra Distance, and distance measures that are based on supervised learning algorithms (see Reis et al. 2018b for details). Section 3.1.1 describes a general distance measure that is based on the Random Forest algorithm, and was shown to work particularly well on astronomical spectra. Since the Random Forest ranks features according to their importance, the Random Forest-based distance is heavily influenced by important features, and is less affected by irrelevant features.



**Figure 7.** Illustration of the disadvantages in using the Euclidean metric to measure distances between astronomical observations. The left panel shows three galaxy spectra, marked by A, B, C, where galaxy A appears visually different. However, the Euclidean distance between galaxy B and C is *larger* than the distance between galaxy A and B, since most of the pixels in a galaxy spectrum are continuum pixels, and galaxy B is slightly bluer than galaxy C. The right panel shows three synthetic light-curves. Since the Euclidean metric is not invariant to horizontal shifts, the distance between light-curve B and C is larger than the distance to light-curve A, although the latter appears visually different.

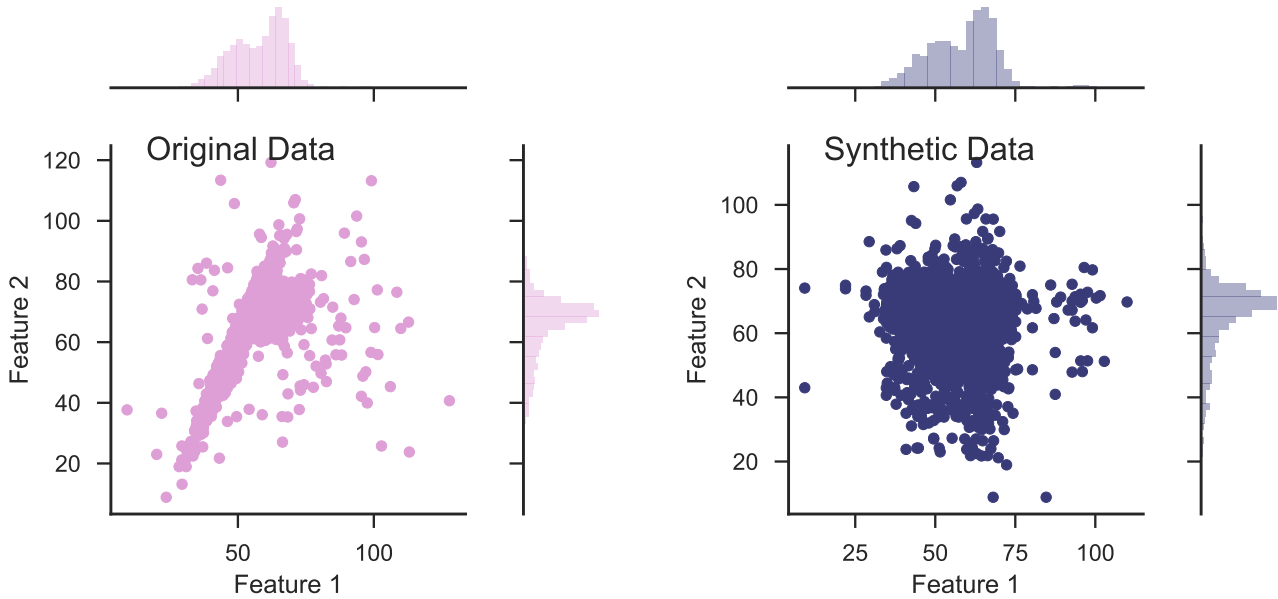
### 3.1.1. General Similarity Measure with Random Forest

So far, the discussion on Random Forest (section 2.4) was focused on a supervised setting, where the model is trained to perform classification and regression. However, Random Forest can be used in an unsupervised setting, to estimate the distance between every pair of objects in the sample. Shi & Horvath (2006) presented such a method, and Baron & Poznanski (2017), followed by Reis et al. (2018a) and Reis et al. (2018b), applied the method to astronomical datasets, with a few modifications that made the algorithm more suitable for such datasets.

In an unsupervised setting, the dataset consists only of measured features, without labels, and is represented by an  $N \times M$  matrix, where every row represents an object ( $N$  objects in the sample), and every column represents a feature ( $M$  measured features per object). To translate the problem into a supervised learning problem that can be addressed with Random Forest, a *synthetic* dataset is built. The synthetic dataset is represented by a  $N \times M$  matrix, similarly to the original dataset, where each fea-

ture (column) in the synthetic data is built by sampling from the marginal distribution of the same feature in the original dataset. One can, instead, shuffle the values in every column of the original data matrix, resulting in a similar synthetic data matrix. The process of creating the synthetic data is illustrated in figure 8 with a simplified example taken from Baron & Poznanski (2017), where the objects in the dataset have only two features. The left panel shows the original dataset, with the marginal distributions in each of the features plotted on the top and on the right. The right panel shows the synthetic dataset, where the features show the same marginal distribution as in the original dataset, but stripped of the covariance seen in the original dataset.

Once the synthetic dataset is constructed, the original dataset is labeled as class 1 and the synthetic dataset is labeled as class 2, and Random Forest is trained to distinguish between the two classes. During this training phase, the Random Forest is trained to detect *covariance*, since it is present only in the original dataset and not in the synthetic one. As a result, the most impor-



**Figure 8.** Illustration of synthetic data construction, taken from Baron & Poznanski (2017), in a simplified example of a dataset with only two features. The left panel shows the distribution of the features of the original dataset, and their marginal distributions. The synthetic dataset (right panel) is constructed by sampling from the marginal distribution of each feature in the original dataset. The resulting dataset shows the same marginal distribution in its features, but is stripped of the covariance that was present in the original dataset.

tant features in the decision trees will be features that show correlations with others. Having the trained forest, the distance between different objects (in the original dataset) is defined as follows. Every pair of objects is propagated through all the decision trees in the forest, and their similarity is defined as the number of times they were both classified as class 1, and reached the *same* terminal leaf. This similarity  $S$  can range between 0 to the number of trees in our forest.  $S$  is a measure of the similarity between these two objects since objects that have a similar path inside the decision tree have similar features, and as a consequence are represented by the same model (for more details see Baron & Poznanski 2017).

Baron & Poznanski (2017), Reis et al. (2018a), and Reis et al. (2018b) showed that this definition of similarity between objects traces valuable information about their different physical

properties. Specifically, they showed that such metric works particularly well for spectra, and traces information coming from different emission and absorption lines, and their connection to the continuum emission. Reis et al. (2018a) applied this method to infrared spectra of stars, and showed that this metric traces physical properties of the stars, such as metallicity, temperature, and surface gravity. The algorithm was implemented in PYTHON and is publicly available on GITHUB<sup>9</sup>.

### 3.2. Clustering Algorithms

Clustering analysis, or clustering, is the task of grouping objects in the sample, such that objects in the same group, which is called a cluster, are more similar to each other than to objects in other groups. The definition of a cluster changes

<sup>9</sup> <https://github.com/dalya/WeirdestGalaxies>



from one algorithm to the next, and this section describes centroid-based clustering (K-means; section 3.2.1) and connectivity-based clustering (Hierarchical clustering; section 3.2.2). Also popular is distribution-based clustering, with the prominent method being Gaussian Mixture Models, which I do not discuss in this document (see e.g., de Souza et al. 2017 and references within). Throughout this section, I give examples with a simplified dataset that consists of two features. Obviously, a two-dimensional dataset can be visualized and clusters can be detected manually, however, these algorithms can be used to search for clusters in complex high-dimensional datasets.

### 3.2.1. *K-means*

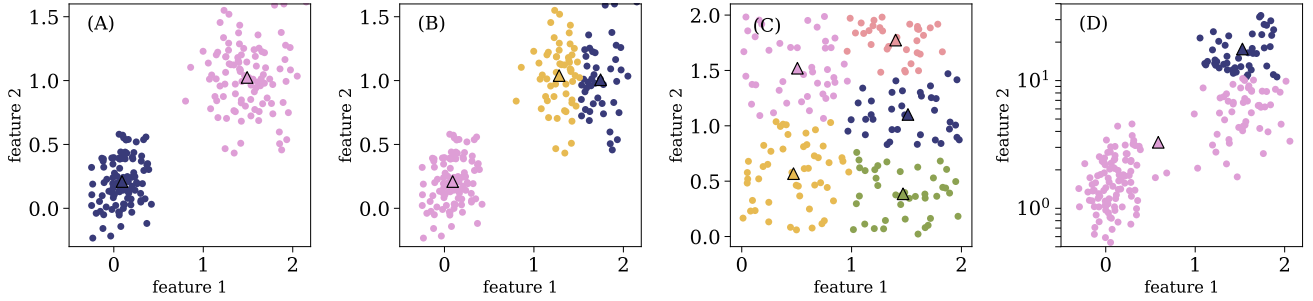
One of the most widely used clustering methods is K-means, which is a centroid-based clustering algorithm (e.g., MacQueen 1967). K-means is simple and robust, even when performing clustering analysis in a high-dimensional space. It was used in Astronomy in various contexts, to study stellar and galaxy spectra, X-ray spectra, solar polarization spectra, spectra from asteroids, and more (see e.g., Balazs et al. 1996; Hojnacki et al. 2007; Galluccio et al. 2008; Sánchez Almeida et al. 2010; Simpson et al. 2012; Sánchez Almeida & Allende Prieto 2013; Garcia-Dias et al. 2018 and references therein).

The first step of K-means is distance assignment between the objects in the sample. The default distance is the euclidean metric, but other metrics, which are more appropriate for the particular dataset at hand, can be used. Then, the algorithm selects  $k$  random objects from the dataset which serve as the initial centroids, where  $k$  is an external free parameter. Each object in the dataset is then assigned to its closest of the  $k$  centroids. Then, new cluster centroids are computed by taking the average position of the objects that are associated with the given cluster. These two steps, re-assigning objects to a cluster according to their distance

from the centroid and recomputing the cluster centroids, are repeated iteratively until reaching convergence. Convergence can be defined in several manners, for example, when the large majority of the objects are no longer reassigned to different centroids (90% and more), or when the cluster centroids converge to a set location. The output of the algorithm consists of the cluster centroids, and an association of the different objects to the different clusters. K-means is available in the SCIKIT-LEARN library<sup>10</sup>.

Panel (A) in Figure 9 shows an application of K-means to a two-dimensional dataset, setting  $k = 2$  and using euclidean distances, where the dots represent the objects in the sample, triangles represent the cluster centroids, and different colors represent different clusters. As noted in the previous paragraph, K-means starts by randomly selecting  $k$  objects as the initial cluster centroids. In some cases, different random assignments of initial centroids might result in different outputs, particularly when K-means converges to a local minimum. To avoid it, one can run K-means several times, each time with different randomly-selected centroids, and select the output that results in the minimum sum of squared distances between the objects and their centroids. Panel (B) in Figure 9 shows an application of K-means to a similar dataset, but with  $k = 3$ .  $k$  is an example of an external parameter that cannot be optimized with the cost function, since the cost function decreases monotonically with  $k$ . Obviously, setting  $k$  to be equal to the number of objects in the sample will result in the minimum possible cost of zero, since each object will be defined as its own cluster. Finding the best  $k$  is not trivial in most of the cases, and studies either use the *elbow method*, or define probability-based scores to constrain it (see e.g., Sánchez Almeida et al. 2010). In

<sup>10</sup> <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>



**Figure 9.** Four examples of K-means application to different two-dimensional datasets, where the objects in the sample are marked with circles and the clusters centroids are marked with triangles. The different colors represent different clusters. Panel (A) shows K-means application with  $k = 2$  and a euclidean distance. Panel (B) shows K-means application to a similar dataset, but with  $k = 3$ . Panel (C) shows K-means output for a dataset without clear clusters, and panel (D) shows the result for a dataset with features which are distributed over different dynamical scales.

some cases, the distribution of the distances of all the objects in the sample contains several distinguishable peaks, and can guide the selection of the correct  $k$ .

Panel (C) in Figure 9 shows an application of K-means to a dataset with no clear clusters. This is an example in which the output consists of clusters while the dataset does not show clear clusters. To test for such cases, one can compare the distribution of distances between objects within the same cluster to the typical distance between cluster centroids. In this particular example, these are roughly similar, suggesting that there are no clear clusters in the dataset. Panel (D) shows an application of K-means to a dataset with features that are distributed over different dynamical scales, and one can see that K-means failed in finding the correct clusters. The K-means cost function is based on the summed distances between the objects and their centroids, and since the distances between the objects are much larger in feature 2 (y-axis), the optimal output is completely dominated by its values. To avoid such issues, it is recommended to either normalize the features, or rank-order them before applying K-means. K-means will also fail when the dataset contains outliers, since they can have a significant impact on the centroids placements,

and therefore on the resulting clusters. Thus, outliers should be removed before applying K-means to the dataset.

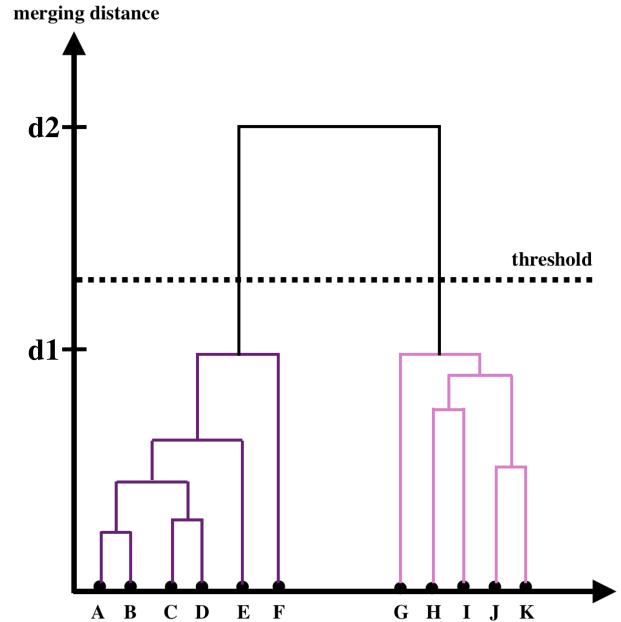
### 3.2.2. Hierarchical Clustering

Hierarchical clustering is another popular algorithm in cluster analysis, aiming at building a hierarchy of clusters (e.g., Ward 1963). The two main types of Hierarchical clustering are Agglomerative Hierarchical clustering, also named the "bottom-up" approach, where each object starts as an individual cluster and clusters are merged iteratively, and Divisive Hierarchical clustering, or "top-down", where all the objects start in one cluster, then split recursively into smaller clusters. Hierarchical clustering has been applied to various astronomical datasets, such as X-ray spectra, extracted features from galaxy images, and absorption spectra of interstellar gas (see e.g., Hojnacki et al. 2007; Baron et al. 2015; Hocking et al. 2015; Peth et al. 2016; Ma et al. 2018a). The discussion in this section is focused on Agglomerative Hierarchical clustering.

The first step of Hierarchical clustering is also assigning distances between the objects in the sample, using the euclidean metric by default. All the objects in the sample start as one-sized clusters. Then, the algorithm merges the two

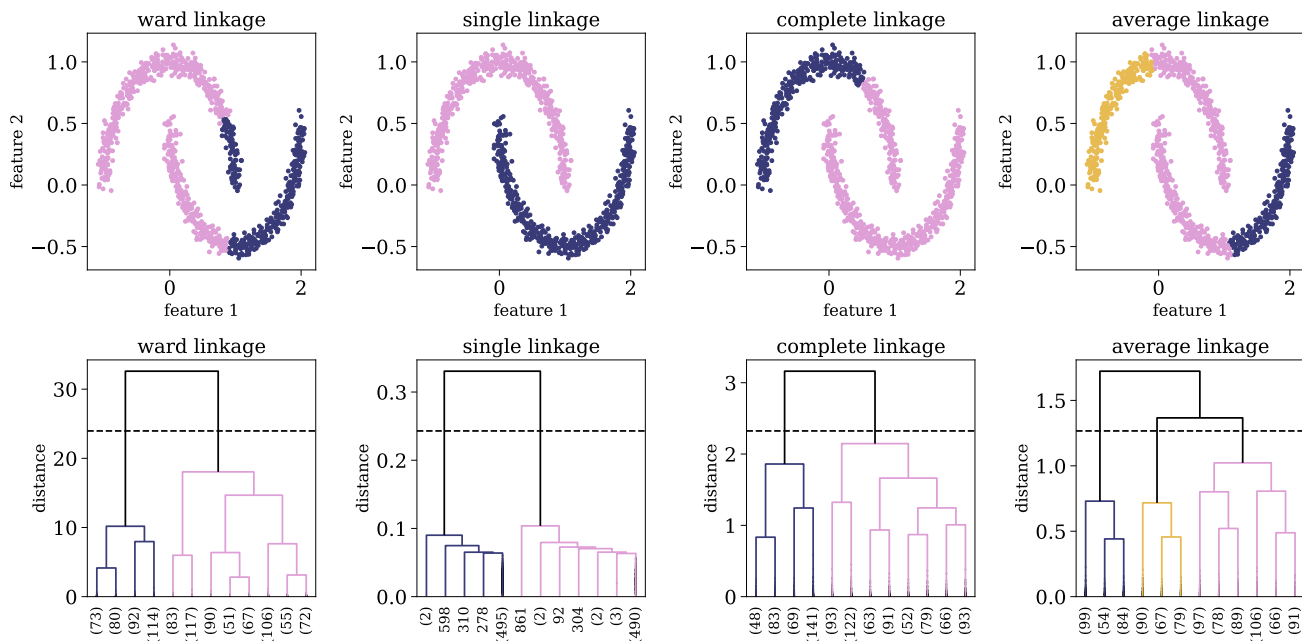
closest clusters into a single cluster. The process is repeated iteratively, merging the two closest clusters into a single cluster, until the dataset consists of a single cluster, which includes many smaller clusters. To do so, one must define a distance between clusters that contain more than one object, also referred to as "linkage method". There are several different linkage methods, which include "complete linkage", where the distance between the clusters is defined as the *maximal* distance between the objects in the two clusters, "single linkage", where the distance between the clusters is defined as the *minimal* distance between the objects in the clusters, "average linkage", where the distance is defined as the *average* distance between the objects in the clusters, and "ward linkage", which minimizes the variance of the clusters merged. The linkage method is an external free parameter of the algorithm, and it has a significant influence on the output.

The result of Hierarchical clustering is usually visualized with a dendrogram. Figure 10 shows an application of Hierarchical clustering to a dataset with 11 objects, marked by A, B, ..., K in the diagram. The dendrogram represents the history of the hierarchical merging process, with the vertical axis showing the distance at which two clusters were merged. Clusters A and B were merged first, since their merging distance is the shortest (up to this point the clusters contain a single object). Then, clusters C and D were merged. Following that, clusters (A, B) and (C, D) were merged, since they were the clusters with the shortest distance. The next two merging clusters are J and K, and the process continues until clusters (A, B, C, D, E, F) and (G, H, I, J, K) are merged into a single cluster. The dendrogram can be used to study the structure of the dataset, in particular, it can be used to infer the number of clusters in the dataset. The example in Figure 10 suggests that the dataset consists of two clusters,



**Figure 10.** Visualization of a dendrogram for a dataset with 11 objects, marked by A, B, ..., K. The dendrogram represents the hierarchical merging history, where the y-axis represents the distance at which two clusters were merged. The clusters that merged with a shorter distance were merged earlier in the process, and in this case, the merger history is A and B, C and D, (A, B) and (C, D), J and K, and so on. The dashed horizontal line represents the threshold  $t$  that is used to define the final clusters in the dataset (see text for more details).

ters, (A, B, C, D, E, F) and (G, H, I, J, K), since the merging distance of inter-cluster objects ( $d1$  in the figure) is much shorter than the merging distance of the two final clusters ( $d2$  in the figure). The dashed horizontal line represents the threshold  $t$  used in Hierarchical clustering for the final cluster definition. Groups that are formed beneath the threshold  $t$  are defined as the final clusters in the dataset, and are the output of the algorithm.  $t$  is an external free parameter of the algorithm, and cannot be optimized using the cost function. It has a significant effect on the resulting clusters, in particular, as  $t$  decreases, the number of resulting clusters increases. In some cases, the dendrogram



**Figure 11.** Application of Hierarchical clustering to a two-dimensional dataset, using different linkage methods, and setting  $t$  to be 0.7 of the maximal merging distance. The top panels show the distribution of the objects in the dataset, colored by the final cluster association. The bottom panels show the resulting dendrograms of each linkage method, and the dashed horizontal line represents the threshold  $t$  used to define the final clusters. The dendrograms are truncated for representational purposes, and the number objects in each truncated branch is indicated on the x-axis in parentheses.

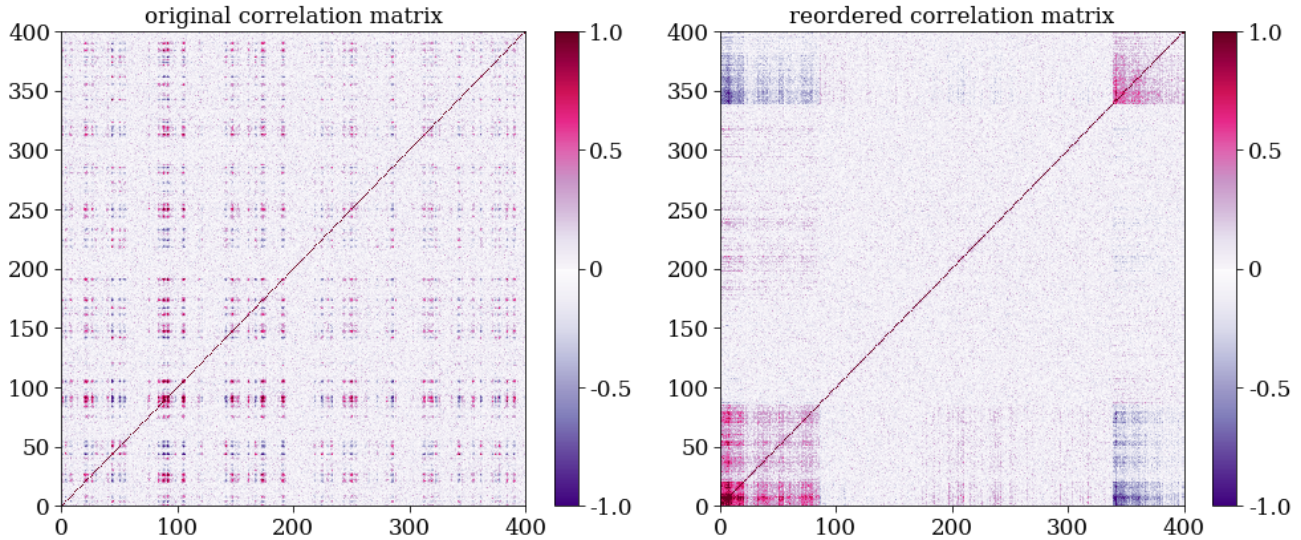
can be used to estimate a "natural" threshold value. Agglomerative Hierarchical clustering is available in `SCIKIT-LEARN`<sup>11</sup>. Visualization of the resulting dendrogram can be done with the `SCIPY` library, and is presented in the hands-on tutorials<sup>12</sup>.

Figure 11 shows an example of Hierarchical clustering application to a two-dimensional dataset, using different linkage methods, setting  $t$  to be 0.7 of the maximal merging distance. The top panels show the distribution of the objects in the dataset, colored by the final clusters detected by the algorithm, and the bottom pan-

els show the dendrograms for each of the linkage methods. Looking at the first row, one can see that different linkage methods result in different cluster definitions, in particular, both the number of detected clusters and the association of objects to clusters change for different linkage methods. In this particular example, it is clear that the single linkage returns the correct output, however, for high-dimensional datasets such a visualization is not possible. To select the "correct" linkage method for the particular dataset at hand, it is advised to examine the resulting dendrograms. In this specific case, the second row shows that the single linkage-based dendrogram reveals the most significant clustering, with two detected clusters. A general rule of thumb to select of the "correct" linkage method is by selecting the linkage that results in the largest difference between the merging

<sup>11</sup> <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html>

<sup>12</sup> [https://github.com/dalya/IAC\\_Winter\\_School\\_2018](https://github.com/dalya/IAC_Winter_School_2018); see also: <https://joernhees.de/blog/2015/08/26/scipy-hierarchical-clustering-and-dendrogram-tutorial/>



**Figure 12.** Application of Hierarchical clustering to reorder and visualize a complex correlation matrix. The left panel shows a correlation matrix calculated for a dataset with 400 objects, where the colorbar represents the Pearson correlation coefficient between every pair of objects. The rows and columns are ordered according to the object index, and little information can be extracted from such a visualization. The right panel shows the same correlation matrix, reordered according to the appearance order of the objects in the dendrogram. The latter representation reveals interesting structures, and can be used to interpret the dataset.

distance of the resulting clusters and the final, maximal, merging distance.

As shown in figure 11, Hierarchical clustering can be used to detect clusters which other clustering algorithms, such as K-means and Gaussian Mixture Models, cannot detect. Since it is based on connectivity, it can be used to detect clusters that are distributed over a non-trivial manifold (this is usually possible only with the single linkage method). Furthermore, Hierarchical clustering is less sensitive to outliers in the dataset, since these will be merged last, and thus will not effect the structure of the dendrogram and the resulting clusters that merged before that. Perhaps the most interesting application of Hierarchical clustering is reordering and visualizing complex distance or correlation matrices. The left panel of Figure 12 shows as example of a correlation matrix, calculated for a complex dataset with 400 objects. The rows and columns of the correlation matrix are ordered according to the object index, and clearly, this

representation conveys little information about the structure of the dataset. Instead, one can perform Hierarchical clustering and extract a dendrogram for this particular dataset. Then, one can rearrange the objects in the correlation matrix according to their order of appearance in the dendrogram. The reordered correlation matrix is shown in the right panel of Figure 12, where one can find at least two clusters of objects, such that objects that belong to a given cluster show strong correlations, and objects that belong to different clusters show a strong negative correlation. Therefore, the process described above may reveal rich structures in the dataset, which may allow one to explore and extract information from it, even without performing cluster analysis (see also [de Souza & Ciardi 2015](#)).

### 3.3. Dimensionality Reduction Algorithms

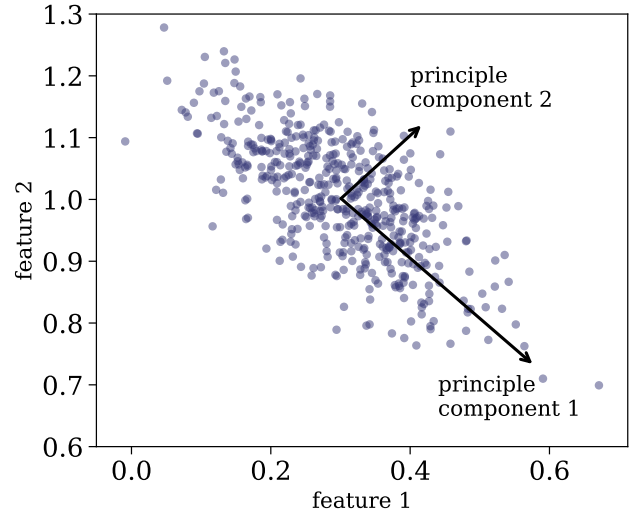
Dimensionality reduction refers to the process of reducing the number of features in the orig-

inal dataset, either by selecting a subset of the features that best describe the dataset, or by constructing a new set of features that provide a good description of the dataset. Some of the dimensionality reduction algorithms provide principle components or prototypes, which are a small set of objects that have the same dimensions as the objects in the original dataset, and are used to represent all the objects in the sample. Other algorithms aim at embedding the high-dimensional dataset onto a low-dimensional space, without using principle components or prototypes. When applying dimensionality reduction algorithms to data, we always lose some information. Our goal is to choose an algorithm that retains most of the *relevant* information, where relevant information strongly depends on our scientific motivation.

Dimensionality reduction is useful for a variety of tasks. In a supervised learning setting, since many algorithms are not capable of managing thousands of features, dimensionality reduction is used to decrease the number of features under consideration, by removing redundancy in the original set of features. Although not very popular in Astronomy, dimensionality reduction is also used for compression, and will become more relevant for surveys such as the SKA (Dewdney et al. 2009), where keeping all the data is no longer possible. Perhaps most importantly, dimensionality reduction can be used to visualize and interpret complex high-dimensional datasets, with the goal of uncovering hidden trends and patterns.

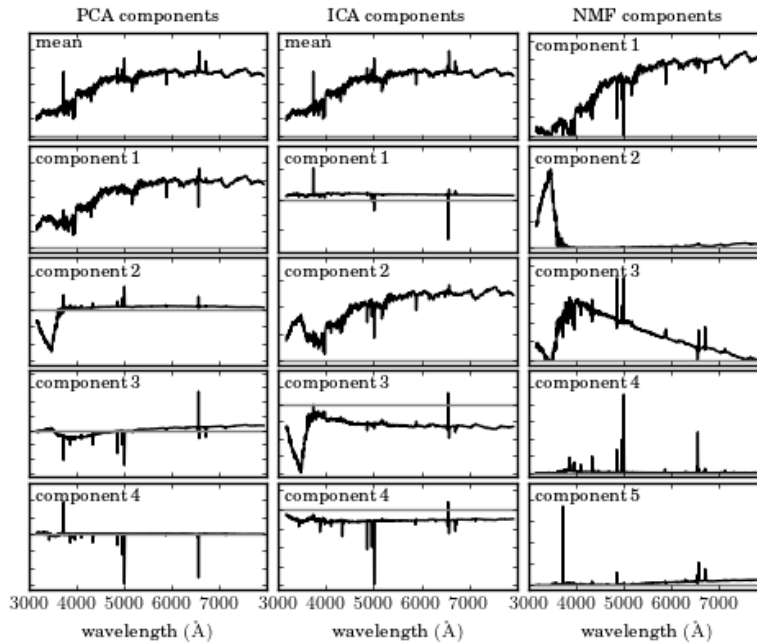
### 3.3.1. Principle Component Analysis

Principle Component Analysis (PCA) is a linear feature projection, which transforms data in high-dimensional space to a lower-dimensional space, such that the variance of the dataset in the low-dimensional representation is maximized. In practice, PCA constructs a covariance matrix of the dataset and computes its eigenvectors. The eigenvectors that correspond



**Figure 13.** Application of PCA to two-dimensional dataset. The two principle components are marked with black arrows, where principle component 1 accounts for most of the variance in the data, and principle component 2, which is orthogonal to principle component 1, accounts for the rest of the variance in the data. Every object in the sample can be accurately represented as a linear combination of the two principle components, and can be represented approximately using the first principle component.

to the largest eigenvalues are used to reconstruct a large fraction of the variance in the original dataset. These eigenvectors, also called principle components, are arranged such that the first principle component has the largest possible variance, and each succeeding component has the highest possible variance, under the constraint that it is orthogonal to the preceding components. The number of principal components is at most the number of features in the dataset. Every object in the sample can be represented by a linear combination of the principle components, where the representation is accurate only when *all* the principle components are used. When a subset of the principle components is used, the representation is approximate, resulting in a dimensionality reduction. The coefficients of the linear combi-



**Figure 14.** Application of PCA, ICA, and NMF to a sample of SDSS spectra, taken from Vanderplas et al. (2012). Additional details on ICA and NMF can be found in Ivezić et al. (2014). The columns represent different algorithms, and the rows represent the resulting components using each of the algorithms, sorted by their importance. The grey horizontal line represents zero flux. While the first two PCA components resemble galaxy spectra (with an old stellar population), the next three components do not represent a physical component in galaxy spectra, in particular, they show negative flux values. On the other hand, the NMF components resemble more physical components, with the first corresponding to an old stellar population, the second corresponding to blue continuum emission, which might be due to an AGN or O- and B-type stars, the third corresponding to younger stellar population (A-type stars), and the fourth and fifth components corresponding to emission lines.

nation of principle components can be used to embed the high-dimensional dataset onto a low-dimensional plane (typically 2 or 3 dimensions). Figure 13 shows an application of PCA to a two-dimensional dataset, where the two principle components are marked with black arrows. One can see that the first principle component is oriented towards the direction with the maximal variance in the data, and the second principle component, which is orthogonal to the first one, describes the remaining variance.

PCA is among the most popular tools in Astronomy, and it has been used to search for multivariate correlations in high-dimensional datasets, estimate physical parameters of systems from their spectra, decompose complex spectra into a set of principle components which

are then used as empirical templates, and more (e.g., Boroson & Green 1992; Djorgovski 1995; Zhang et al. 2006; Vanden Berk et al. 2006; Rogers et al. 2007; Re Fiorentin et al. 2007; Bailey 2012). It is simple to use, has no free parameters, and is easily interpretable. However, PCA performs a *linear* decomposition of the objects in the sample, which is not appropriate in many contexts. For example, absorption lines and dust extinction are multiplicative effects which cannot be described by a linear decomposition. Furthermore, PCA tends to find linear correlations between variables, even if those are non-linear, and it fails in cases where the mean and the covariance are not enough to define the dataset. Since it constructs its principle components to trace the maximal variance in

the data, it is extremely sensitive to outliers, and these should be removed prior to applying PCA. Finally, the principle components of the dataset can contain negative values, which is also not appropriate in many astronomical setups. For example, applying PCA to galaxy spectra results in principle components with negative values, which is of course not physical, since the emission of a galaxy is a sum of *positive* contributions of different light sources, *attenuated* by absorbing sources such as dust or gas.

Finally, it is worth noting two additional techniques, Independent Component Analysis (ICA; Hyvärinen & Oja 2000) and Non-Negative Matrix Factorization (NMF; Paatero & Tapper 1994), which are useful for a variety of tasks. ICA is a method used to separate a multivariate signal into additive components, which are assumed to be non-Gaussian and statistically independent from each other. ICA is a very powerful technique, which is often invoked in the context of blind signal separation, such as the "cocktail party problem". NMF decomposes a matrix into the product of two non-negative ones, and is used in Astronomy to decompose observations to non-negative components. Figure 14 shows an application of PCA, ICA, and NMF, taken from Vanderplas et al. (2012), on SDSS spectra (see Ivezić et al. 2014 for additional details about ICA and NMF). The columns represent different algorithms, and the rows represent the resulting components using each of the algorithms, sorted by their importance. One can see that while the first two PCA components resemble galaxy spectra (with an old stellar population), the next three components do not represent a physical component in galaxy spectra, in particular, they show negative flux values. On the other hand, the NMF components resemble more physical components, with the first corresponding to an old stellar population, the second corresponding to

blue continuum emission, which might be due to an AGN or O- and B-type stars, the third corresponding to younger stellar population (A-type stars), and the fourth and fifth components corresponding to emission lines. Obviously, the resemblance is not perfect and one can see residual emission line components in the first and third components (in the first component these are described by absorption lines with central wavelengths corresponding to the strongest emission lines in galaxy spectra). The three algorithms are available in SCIKIT-LEARN<sup>13</sup>.

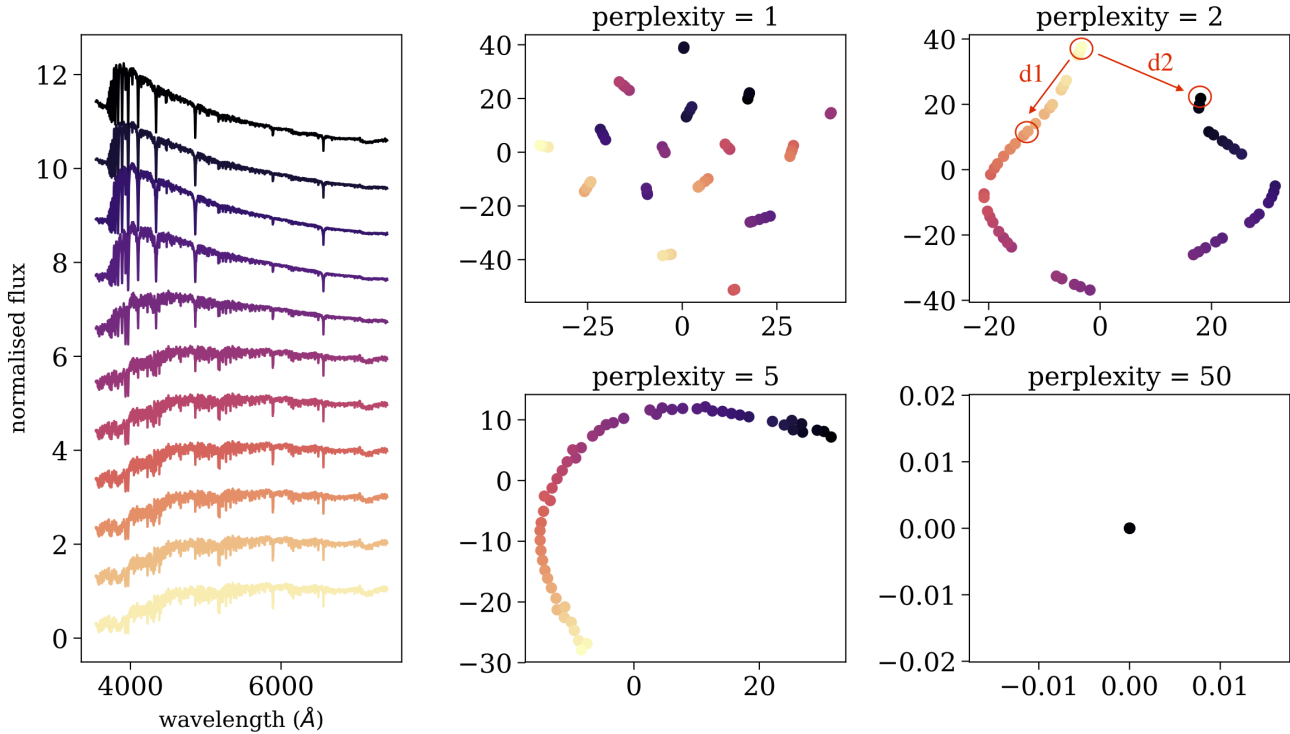
### 3.3.2. *t-Distributed Stochastic Neighbor Embedding*

*t*-Distributed Stochastic Neighbor Embedding (tSNE; van der Maaten & Hinton 2008) is a non-linear dimensionality reduction technique that embeds high-dimensional data in a low dimensional space, typically of two or three dimensions, and is mostly used to visualize complex datasets. The algorithm models every high-dimensional object using a two (or three) dimensional point, such that similar objects are represented by nearby points, whereas dissimilar objects are represented by distant points, with a high probability. tSNE has been used in Astronomy to visualize complex datasets and distance matrices, and study their structure (e.g., Lochner et al. 2016; Anders et al. 2018; Nakoneczny et al. 2018; Reis et al. 2018a; Alibert 2019; Giles & Walkowicz 2019; Moreno et al. 2019).

The first step of tSNE is to assign distances between the objects in the sample, using the euclidean metric by default. Then, tSNE constructs a probability distribution over pairs of

<sup>13</sup> <https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>  
<https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.FastICA.html>  
<https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html>





**Figure 15.** Application of tSNE to a sample of 53 synthetic single stellar population models, with different ages. The left panel shows 11 out of the 53 spectra, where each spectrum consists of 4300 flux values. The right panels show the resulting two-dimensional embedding using different perplexity values, where every point in the 2D plane is colored according to the stellar age, such that purple points represent young stellar populations, and yellow points represent old stellar populations.

high-dimensional objects such that similar objects have a high probability of being picked, whereas dissimilar objects have an extremely low probability of being picked. This is achieved by modeling the probability distribution using a Gaussian kernel, which depends on the assigned distance between the objects, and a scale parameter, named the *perplexity*, which is an external free parameter of the algorithm. The perplexity affects the neighborhood of objects being considered, in terms of their probability of being selected, where a small perplexity results in a very small neighborhood around a given object, and a large perplexity results in a larger neighborhood. The algorithm then embeds the high-dimensional objects into a low-dimensional space, such that the probability distribution over pairs of points in the low-dimensional plane

will be as similar as possible to the probability distribution in the high-dimensional space. The axes in the low-dimensional representation are meaningless and not interpretable. tSNE is available in the SCIKIT-LEARN library<sup>14</sup>.

Figure 15 shows an application of tSNE to a sample of 53 single stellar population models, taken from the MILES library (Vazdekis et al. 2010). The left panel shows 11 out of the 53 spectra, where each spectrum has 4300 flux values, and therefore 4300 features, ordered by age. While the dataset appears complex, it actually represents a one-dimensional manifold, where all the observed properties can be attributed to a change in a single parameter,

<sup>14</sup> <https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html>

the age. Therefore, we expect that in the low dimensional representation, all the objects will occupy a single line. The right panels show the tSNE two-dimensional embedding for different perplexity values. When the perplexity is set to  $p = 1$ , the neighborhoods considered by tSNE are too small, and the data is represented by many small clusters, although the dataset does not consist of clusters. When the perplexity is set to  $p = 2$  or  $p = 5$ , the dataset is represented by an almost perfect one-dimensional manifold, tracing the correct structure of the data. When the perplexity is too high, e.g.  $p = 50$ , all the points are plotted at the origin, and no structure can be seen. The perplexity has a significant effect on the resulting embedding, and cannot be optimized using the tSNE cost function. It represents the scale of the clusters tSNE is sensitive to, and setting a small perplexity value allows detection of small clusters, while setting a large perplexity value allows study of the global structure of the dataset. However, one must take into account that the low-dimensional embeddings by tSNE sometimes show small clusters, even if these do not exist in the dataset. In addition, one must be careful when trying to interpret long distances in the tSNE map. For example, when  $p = 2$ , tSNE embeds the objects considering only neighborhoods of up to about the 6 nearest neighbors around each point, thus, distances smaller than those spanned by this neighborhood are conserved, while longer distances are not. While the distances  $d1$  and  $d2$  appear similar in the tSNE map in Figure 15, the distances between the corresponding objects in the original distance matrix are different by a factor of more than five.

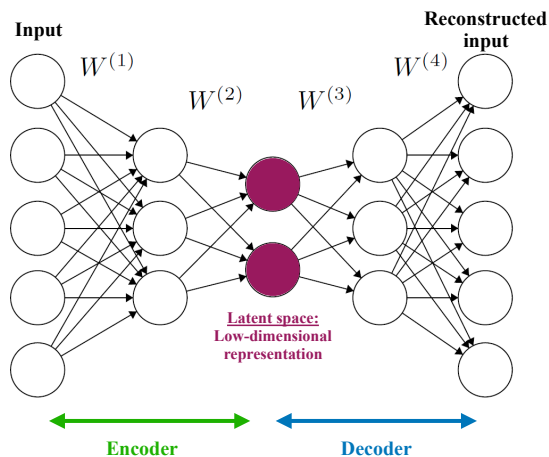
Finally, it is worth noting an additional tool, Uniform Manifold Approximation and Projection (UMAP; McInnes et al. 2018), which can be used to perform non-linear dimensionality reduction. UMAP is a fast algorithm, that supports a wide variety of distance metrics, includ-

ing non-metric distance functions such as cosine distance and correlation distance. Furthermore, McInnes et al. (2018) show that UMAP often outperforms tSNE at preserving global structures in the input dataset. UMAP is implemented in PYTHON and is publicly available on GITHUB<sup>15</sup>.

### 3.3.3. Autoencoders

An autoencoder is a type of artificial neural network used to learn an efficient low-dimensional representation of the input dataset, and is used for compression, dimensionality reduction, and visualization (Gianniotis et al. 2015; Yang & Li 2015; Gianniotis et al. 2016; Ma et al. 2018b; Schawinski et al. 2018). Figure 16 shows a simplified example of an autoencoder architecture. The network consists of two parts, the *encoder* and the *decoder*. In the encoding stage, the input data propagates from the input layers to the hidden layers, which typically have a decreasing number of neurons, until reaching the bottleneck, which is the hidden layer consisting of two neurons. In other words, the encoder performs a compression of the input dataset, by representing it using two dimensions. Then, the decoder reconstructs the input data, using the information from the two-dimensional hidden layer. The weights of the network are optimized during the training, where the loss function is defined as the squared difference between the input data and the reconstructed data. Once trained, the dataset can be represented in a low-dimensional space, also called the latent space, using the values given in the bottleneck hidden layer. As for all neural network-based algorithms, these networks are general and flexible, and can be used to represent very complex datasets. However, this complexity comes with a price. Such networks can have many different architectures,

<sup>15</sup> <https://github.com/lmcinnes/umap>



**Figure 16.** A simplified example of an autoencoder architecture, used to perform compression, dimensionality reduction, and visualization. The network consists of two parts, the *encoder* and the *decoder*. The encoder reduces the dimensions of the input data, while the decoder reconstructs the input using the low-dimensional representation. The weights of the network are optimized during training to minimize the squared differences between the input and its reconstruction. The bottleneck of the network, also called the latent vector or latent space, represents the low-dimensional representation of the input dataset.

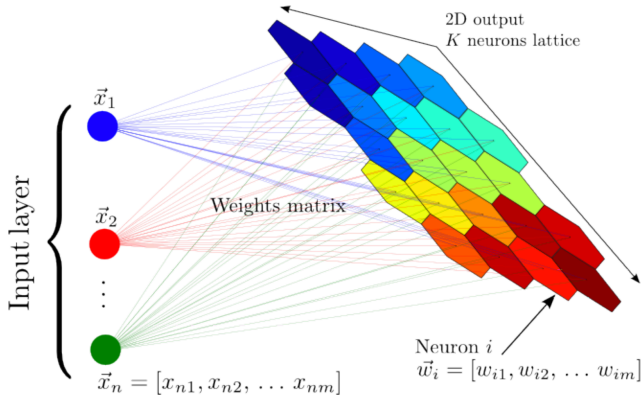
and thus a large number of free parameters, and are difficult to interpret (see lectures by M. Huertas-Company for additional details).

### 3.3.4. Self Organizing Maps

A self-organizing map (also named Kohonen map; Kohonen 1982) is a type of artificial neural network that is trained in an unsupervised manner and produces a low-dimensional (typically two-dimensional) representation of the input dataset. During training, the two-dimensional map self-organizes itself to match the input dataset, preserving its topology very closely. In Astronomy, self-organizing maps have been used to perform semi-supervised classification and regression, clustering, visualization of complex datasets, and outlier detection (see e.g., Meusinger et al. 2012; Fustes et al. 2013; Car-

rasco Kind & Brunner 2014; Armstrong et al. 2016; Polsterer et al. 2016; Armstrong et al. 2017; Meusinger et al. 2017; Rahmani et al. 2018). Figure 17 shows a schematic illustration of a self-organizing map, taken from Carrasco Kind & Brunner (2014). The input dataset consists of  $n$  objects with  $m$  features each. The network consists of an input layer with  $m$  neurons, and an output layer with  $k$  neurons, organized as a two-dimensional lattice. The neurons from the input layer are connected to the output layer with weight vectors, which have the same dimensions as the input objects ( $m$  in this case). Contrary to typical artificial neural networks, where the weights are used to multiply the input object values, followed by an application of an activation function, the weights of self-organizing maps are characteristics of the output neurons themselves, and they represent the "coordinates" of each of the  $k$  neurons in the input data space. That is, the weight vectors serve as templates (or prototypes) of the input dataset.

The training of a self-organizing map is a competitive process, where each neuron in the output layer competes with the other neurons to best represent the input dataset. The first step of self-organizing maps is a random initialization of the weights, where typically, the initial weights are set to be equal to randomly-selected objects from the input dataset. Then, the algorithm iterates over the objects in the input dataset. In each iteration, the algorithm computes the distance between the particular object and all the neurons in the output layer, using the euclidean distance between the object's features and the weight vectors that are associated with the neurons. Then, the algorithm determines the closest neuron to the input object, and updates its weight vector to be somewhat closer to the input object. The algorithm also updates the weight vectors of the neighboring neurons, such that closer neurons are updated



**Figure 17.** A schematic illustration of a self-organizing map, taken from Carrasco Kind & Brunner (2014). The input dataset consists of  $n$  objects, each with  $m$  features, and it is mapped to a two-dimensional lattice of  $k$  neurons. Each neuron is represented by a weight vector, which has the same dimensions as the input objects. The weights are characteristics of the neurons themselves, and they represent the coordinate of each neuron in the input data space. These weights are updated during the training process, and once the algorithm is trained, they represent a set of templates (or prototypes) that describe the different objects in the dataset.

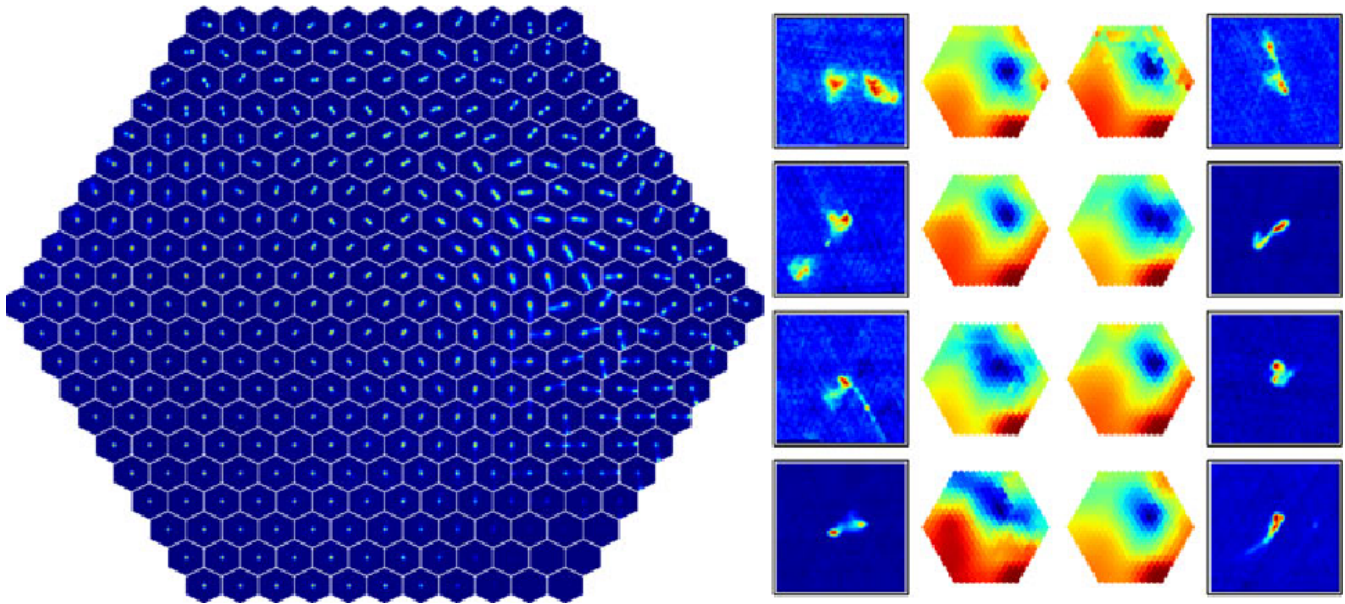
more than farther neurons. The update magnitude is determined by a kernel function, usually a Gaussian, which depends on a *learning radius* parameter. The update magnitude of all the neurons depends on a *learning rate* parameter, where both the *learning radius* and the *learning rate* decrease with time. The self-organizing map converges after a number of iterations, and in its final form it separates the input dataset into groups of similar objects, which are represented by nearby neurons in the output layer. The final weights of the network represent prototypes of the different groups of objects, and they are usually used to manually inspect the dataset. Self-organizing map is implemented in PYTHON and is publicly available on GITHUB<sup>16</sup>.

<sup>16</sup> <https://github.com/sevamoo/SOMPY>

Self-organizing maps are general and flexible, and their capability of sorting the input dataset onto a two-dimensional plane allows manual inspection of a relatively small number of prototypes, and use these to explore the structure of the dataset. However, in their simpler versions, self-organizing maps cannot be applied to astronomical images, since the algorithm is based on euclidean similarities, which is not invariant to rotations and flips. Polsterer et al. (2016) developed PINK, which is a self-organizing map that is invariant to rotations and flips, which is particularly useful for galaxy images, for example. Figure 18 shows an application of PINK to 200 000 radio images, taken from Polsterer et al. (2016). The left panel shows the resulting two-dimensional map, which contains the derived prototypes. These prototypes allow a clear separation of the input dataset into different morphological types, and by manually inspecting the map, one can explore the range of morphological types in the dataset, without manually inspecting thousands of images. The right panel shows eight outliers, which are objects which are not well-represented by any prototype.

### 3.4. Anomaly Detection

Outlier detection is the natural step after classification and the analysis of the dataset structure. In Astronomy, outlier detection algorithms were applied to various datasets, including galaxy and stellar spectra, galaxy images, astronomical light-curves such as variable stars, radio images, and more (Meusinger et al. 2012; Protopapas et al. 2006; Fustes et al. 2013; Nun et al. 2016; Agnello 2017; Baron & Poznanski 2017; Solarz et al. 2017; Hocking et al. 2018; Reis et al. 2018a; Segal et al. 2018; Giles & Walkowicz 2019). There are various types of outliers we expect to find in our datasets. Outliers can be objects that were not intended to be in our datasets, such as a quasar in a sample of stars, or various observational or pipeline errors.



**Figure 18.** Application of PINK to 200 000 radio images from Radio Galaxy Zoo, taken from Polsterer et al. (2016). The left panel shows the resulting two-dimensional map containing the derived prototypes. The right panel shows eight outliers that were selected based on their dissimilarity with the prototypes, and heatmaps that indicate their distance to all the prototypes.

Such outliers are not scientifically interesting, but we wish to detect them and remove them from our samples. Outliers can be extreme objects drawn from the tail of well-characterized distributions, such as the most massive supermassive black hole or the most luminous supernova. Such objects are interesting because they allow us to test our models, which were built to describe the bulk of the population, in extreme regimes, and by studying them, we can learn more about the bulk of the population. The most interesting types of the outliers are the "unknown unknowns" (Baron & Poznanski 2017), objects we did not know we should be looking for, and may be completely new objects which offer the opportunity to unveil new physics. Furthermore, in Astronomy, outliers can actually be very common phenomena, that occur on time-scales much shorter than other time scales of the system. For example, if every galaxy in the universe becomes green for 100 years, while in the rest of the time it evolves on a time scale of hundreds of million of years with its "regular" blue or red color, the proba-

bility of detecting a galaxy in its "green phase" using surveys such as the SDSS is close to zero. Therefore, although this "green phase" occurs in every galaxy and might be a fundamental phase in galaxy evolution, green galaxies will appear as outliers in our datasets.

Unknown unknowns are usually detected serendipitously, when experts visually inspect their datasets, and discover an object that does not follow the accepted paradigm. Manual inspection becomes impractical in the big data era, where surveys provide millions of observations of a particular class of objects. Indeed, the vast majority of observations are no longer inspected by humans. To facilitate new discoveries, we must develop and use off-the-shelf algorithms to perform anomaly detection (see discussion by Norris 2017a,b). Outlier detection is in some sense the ultimate unsupervised learning task, since we cannot define what we are looking for. Therefore, outlier detection algorithms must be as generic as possible, but at the same time they must be optimized to learn the characteristics of the dataset at hand, since

outliers are defined as being different, in some sense, from the bulk of the population.

In most supervised and unsupervised tasks, the input dataset either consists of the original astronomical observations (spectra, light-curves, images, etc), or features that were extracted from the original observations, and there are advantages and disadvantages to both of these choices. However, anomaly detection should be carried out on a dataset which is as close as possible to the original dataset, and not on extracted features. First, defining features directly limits the type of outliers one might find. For example, in galaxy spectra, extracted features include the properties of the stellar population, and measurements of different emission lines. Such features will not carry information about new unidentified emission lines in a galaxy, and if such anomalous galaxies exist, they will not be marked as outliers. Second, extracted features are usually an output of some pipeline, which can sometimes fail and extract erroneous feature measurements. Such errors typically occur in outlying objects, since the models that are used to extract the features cannot describe them well. In such cases, the resulting features, which were wrongly measured, typically show values that are consistent with features measured for the bulk of the population, and such outliers cannot be detected (see [Baron & Poznanski 2017](#) for examples).

#### 3.4.1. *Anomaly Detection with Supervised Learning*

Supervised learning algorithms can be used to detect outliers, in both classification and regression tasks. When applied to new previously unseen objects, most supervised learning algorithms provide some measure of uncertainty or classification probability. Outliers can be defined as objects that have a large classification uncertainty or low classification probability. For example, a Random Forest algorithm is trained to distinguish between the spectra of stars and

quasars. Then, it predicts the class (and class probabilities) of previously unseen objects. An object that is classified as a star with a probability of 0.55 (and probability of 0.45 to be a quasar) is probably more anomalous than an object that is classified as a star with a probability of 0.95. Therefore, the anomaly score of the different objects in the sample can be defined according to the classification probability. While easy to interpret, such a process will only reveal the outliers that "shout the loudest". Using again the star-quasar classification example, while a galaxy spectrum will be marked as an outlier by such a procedure, more subtle outliers, such as stars with anomalous metallicity, will not be detected. This is because supervised learning algorithms are optimized to perform classification (or regression), and as such, they will use only the features that are relevant for the classification task at hand. Therefore, objects which show outlier properties in these features will be detected, while objects that show outlier properties in features that are less relevant for the classification task, will not be detected.

#### 3.4.2. *Anomaly Detection with Unsupervised Learning*

There are several ways to perform outlier detection in an unsupervised setting. First, one can assign pair-wise distances between the objects in the sample, and define outliers as objects that have a large average distance from the rest of the objects (see e.g., [Protopapas et al. 2006](#); [Baron & Poznanski 2017](#)). As discussed in section 3.1, in some cases, using a euclidean metric might not result in an optimal performance, and one should consider other metrics. For example, the unsupervised Random Forest-based distance was shown to work particularly well on spectra ([Baron & Poznanski 2017](#); [Reis et al. 2018a,b](#)), and cross correlation-based distances work well for time series ([Protopapas et al. 2006](#); [Nun et al. 2016](#)).

An additional way to perform outlier detection is by applying dimensionality reduction (which sometimes requires distance assignment as well). Once the high dimensional dataset is embedded onto a low dimensional plane, it can be visualized. Outliers can be defined as objects that are located far from most of the objects or on the edges of the observed distributions. The success of this process strongly depends on the procedure used to perform dimensionality reduction, and one must take into account the internal choices and loss function of the algorithm. For example, when using PCA to project the dataset onto a two-dimensional plane, one must take into account that while some outliers will be objects with extreme values in this 2D projection, and thus will be detected, other outliers can be objects that show extreme values in the other eigenvectors, which are not used for the projection and visualization, and thus will not show up as outliers. Another example is an auto-encoder, where some outliers will show up as extreme objects in the latent space (the two-dimensional representation), while other outliers will show typical values in the latent space, but a large reconstruction error on the decoder side. The final example is tSNE, where one must take into account the fact that the distances of the objects in the two-dimensional projection are not euclidean. In particular, while short distances in the original distance matrix are roughly conserved in the tSNE map, long distances are not.

### 3.4.3. One-Class SVM

One of the most popular outlier detection algorithms is one-class SVM (Schölkopf et al. 1999). In one-class SVM, the input data is considered to be composed of a single class, represented by a single label, and the algorithm estimates a distribution that encompasses most of the observations. This is done by estimating a probability distribution function which makes most of the observed data more likely than the

rest, and a decision rule that separates these observations by the largest possible margin. This process is similar to the supervised version of SVM, but applied to a dataset with a single label. To optimize over the free parameters of SVM, such as the kernel shape and its parameters, the input dataset is usually divided into a training set and a validation set (there is no need for a test set, since this is an unsupervised setting). The algorithm is trained on the training set, resulting in some decision function, while the free parameters are optimized using the validation set. Therefore, the chosen kernel shape and its free parameters are chosen to give the highest classification accuracy on the validation set, where the classification accuracy is defined by the number of objects that are classified as inliers (the opposite of outliers) by the resulting decision function (see e.g., Solarz et al. 2017). The learned decision function is then used to define outliers. Outliers are objects that are outside the decision function, and their anomaly score can be defined by the distance of the outliers from the decision function.

One-class SVM is feasible only with datasets composed of a handful of features. Therefore, it cannot be directly applied to astronomical observations such as images, light-curves, or spectra, but can be applied to photometry or derived features. One-class SVM is available in the SCIKIT-LEARN library<sup>17</sup>.

### 3.4.4. Isolation Forest

Another very popular outlier detection algorithm is Isolation Forest (Liu et al. 2008). Isolation Forest consists of a set of random trees. The process of building such a forest is similar to the training process of Random Forest, but here both the feature and the splitting value are *randomly* selected at each node. Within each tree, outliers will tend to separate earlier from the

<sup>17</sup> <https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html>

rest of the sample. Therefore, the anomaly score can be defined as the depth at which a specific object was split from the rest, averaged over all the trees in the forest. The running time of Isolation Forest is  $O(N)$ , where  $N$  is the number of objects in the sample, and it can be applied to datasets with numerous features. [Baron & Poznanski \(2017\)](#) compared its performance to that of the unsupervised Random Forest-based outlier detection, and found that Isolation Forest is capable of finding the most obvious outliers, those that "shout the loudest", but cannot detect subtle outliers, which are typically more interesting in an astronomical context. On the other hand, [Reis et al. \(2018a\)](#) and [Reis et al. \(2018b\)](#) found that when tuning the range of possible feature values that are randomly selected in each node (i.e., instead of defining the possible range to be between the minimum and maximum feature values, one could define the range to be between the 10th and 90th percentiles), Isolation Forest results in a comparable performance to that of the unsupervised Random Forest. Isolation Forest is available in the SCIKIT-LEARN library<sup>18</sup>.

#### 4. SUMMARY

In recent years, machine learning algorithms have gained increasing popularity in Astronomy, and have been used for a wide variety of tasks. In this document I summarized some of the popular machine learning algorithms and their application to astronomical datasets. I reviewed basic topics in supervised learning, in particular selection and preprocessing of the input dataset, evaluation metrics of supervised algorithms, and a brief description of three popular algorithms: SVM, Decision Trees and Random Forest, and shallow Artificial Neural Networks. I mainly focused on unsupervised learn-

ing techniques, which can be roughly divided into clustering analysis, dimensionality reduction, and outlier detection. The most popular application of machine learning in Astronomy is its supervised setting, where a machine is trained to perform classification or regression according to previously-acquired scientific knowledge. While less popular in Astronomy, unsupervised learning algorithms can be used to mine our datasets for novel information, and potentially enable new discoveries. In section 4.1 I list a number of open questions and issues related to the application of machine learning algorithms in Astronomy. Then, in section 4.2, I refer the reader to textbooks and online courses that give a more extensive overview of the subject.

#### 4.1. Open Questions

The main issues of applying supervised learning algorithms to astronomical datasets include uncertainty treatment, knowledge transfer, and interpretability of the resulting models. As noted in section 3.1.1, most supervised learning algorithms are not constructed for astronomical datasets, and they implicitly assume that all measured features are of the same quality, and that the provided labels can be considered as ground truth. However, astronomical datasets are noisy and have gaps, and in many cases, the labels provided by human experts suffer from some level of ambiguity. As a result, supervised learning algorithms perform well when applied to high signal-to-noise ratio datasets, or to datasets with uniform noise properties. The performance of supervised learning algorithms strongly depends on the noise characteristics of the objects in the sample, and as such, an algorithm that was trained on a dataset with particular noise characteristics will fail to generalize to a similar dataset with different noise characteristics. It is therefore necessary to change existing tools and to develop new algorithms, which take into account uncertainties in the

<sup>18</sup> <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html>



dataset during the model construction. Furthermore, such algorithms should provide prediction uncertainties, which are based on the intrinsic properties of the objects in the sample and on their measurement uncertainties.

The second challenge in applying supervised learning algorithms to astronomical datasets is related to knowledge transfer. That is, an algorithm that is trained on a particular survey, with a particular instrument, cadence, and object targeting selection, will usually fail to generalize to a different survey with different characteristics, even if the intrinsic properties of the objects observed by the two surveys are similar. As a result, machine learning algorithms are typically applied to concluded surveys, and rarely applied to ongoing surveys that have not yet collected enough labeled data. The topic of knowledge transfer is of particular importance when searching for rare phenomena, such as gravitational lenses in galaxy images, where supervised learning algorithms that are trained on *simulated* data cannot generalize well to real datasets. This challenge can be addressed with *transfer learning* techniques. While such techniques are discussed in the computer science literature, they are seldom applied in Astronomy.

The third challenge in applying supervised learning algorithms to astronomical datasets is related to the interpretation of the resulting models. While supervised learning algorithms offer an extremely flexible and general framework to construct complex decision functions, and can thus outperform traditional algorithms in classification and regression tasks, the resulting models are often difficult to interpret. That is, we do not always understand *what* the model learned, and *why* it makes the decisions that it makes. As scientists, we usually wish to understand the constructed model and the decision process, since this information can teach us something new about the underlying physics. This challenge is of particular importance in

state-of-the-art deep learning techniques, which were shown to perform exceptionally-well in a variety of tasks. As we continue to develop new complex tools to perform classification and regression, it is important to devise methods to interpret their results as well.

When applying unsupervised learning algorithms to astronomical datasets, the main challenges include the interpretation of the results and comparison of different unsupervised learning algorithms. Unsupervised learning algorithms often optimize some internal cost function, which does not necessarily coincide with our scientific motivation, and since these algorithms are not trained according to some definition of "ground truth", their results might lead to erroneous interpretations of trends and patterns in our datasets. Many of the state-of-the-art algorithms are modular, thus allowing us to define a cost function that is more appropriate for the task at hand. It is therefore necessary to formulate cost functions that match our scientific goals better. To interpret the results of an unsupervised learning algorithm and to compare between different algorithms, we still use domain knowledge, and the process cannot be completely automatized. To improve the process of interpreting the results, we must improve the machine-human interface through which discoveries are made, e.g., by constructing visualization tools that incorporate post-processing routines which are typically carried out after applying unsupervised learning algorithms. Finally, as we continue to apply unsupervised learning algorithms to astronomical datasets, it is necessary to construct evaluation metrics that can be used to compare the outputs of different algorithms.

#### 4.2. *Further Reading*

To learn more about the basics of machine learning algorithms, I recommend the publicly-available machine learning course in COURSE-

ERA<sup>19</sup>. For an in-depth reading on statistics, data mining, and machine learning in Astronomy, I recommend the book by Ivezic et al. (2014), which covers in greater depth many of the topics presented in this document, and many other related topics. For additional examples on machine learning in Astronomy, implemented in PYTHON, I recommend astroML<sup>20</sup> (Vanderplas et al. 2012).

I am grateful to I. Arcavi, N. Lubelchick, D. Poznanski, I. Reis, S. Shahaf, and A. Stern-

berg for valuable discussions regarding the topics presented in this document and for helpful comments on the text.

*Software:* astroML (Vanderplas et al. 2012), scikit-learn (Pedregosa et al. 2011), SciPy (Jones et al. 2001–), matplotlib (Hunter 2007), and IPython (Pérez & Granger 2007).

## REFERENCES

- Agnello, A. 2017, MNRAS, 471, 2013
- Alibert, Y. 2019, arXiv e-prints, arXiv:1901.09719
- Anders, F., Chiappini, C., Santiago, B. X., et al. 2018, A&A, 619, A125
- Armstrong, D. J., Pollacco, D., & Santerne, A. 2017, MNRAS, 465, 2634
- Armstrong, D. J., Kirk, J., Lam, K. W. F., et al. 2016, MNRAS, 456, 2260
- Ascasibar, Y., & Sánchez Almeida, J. 2011, MNRAS, 415, 2417
- Bailey, S. 2012, PASP, 124, 1015
- Balazs, L. G., Garibjanyan, A. T., Mirzoyan, L. V., et al. 1996, A&A, 311, 145
- Ball, N. M., & Brunner, R. J. 2010, International Journal of Modern Physics D, 19, 1049
- Banerji, M., Lahav, O., Lintott, C. J., et al. 2010, MNRAS, 406, 342
- Baron, D., & Poznanski, D. 2017, MNRAS, 465, 4530
- Baron, D., Poznanski, D., Watson, D., et al. 2015, MNRAS, 451, 332
- Bellm, E. 2014, in The Third Hot-wiring the Transient Universe Workshop, ed. P. R. Wozniak, M. J. Graham, A. A. Mahabal, & R. Seaman, 27–33
- Bilicki, M., Hoekstra, H., Brown, M. J. I., et al. 2018, A&A, 616, A69
- Blake, C., Collister, A., Bridle, S., & Lahav, O. 2007, MNRAS, 374, 1527
- Bloom, J. S., Richards, J. W., Nugent, P. E., et al. 2012, PASP, 124, 1175
- Boroson, T. A., & Green, R. F. 1992, ApJS, 80, 109
- Breiman, L. 2001, Machine Learning, 45, 5. <http://dx.doi.org/10.1023/A:1010933404324>
- Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. 1984, Monterey, CA: Wadsworth & Brooks/Cole Advanced Books & Software
- Brescia, M., Cavuoti, S., D’Abrusco, R., Longo, G., & Mercurio, A. 2013, ApJ, 772, 140
- Brescia, M., Cavuoti, S., Longo, G., & De Stefano, V. 2014, A&A, 568, A126
- Brescia, M., Cavuoti, S., Paolillo, M., Longo, G., & Puzia, T. 2012, MNRAS, 421, 1155
- Carliles, S., Budavári, T., Heinis, S., Priebe, C., & Szalay, A. S. 2010, ApJ, 712, 511
- Carrasco Kind, M., & Brunner, R. J. 2014, MNRAS, 438, 3409
- Castro, N., Protopapas, P., & Pichara, K. 2018, AJ, 155, 16
- Collister, A. A., & Lahav, O. 2004, PASP, 116, 345
- Connolly, A. J., Szalay, A. S., Bershady, M. A., Kinney, A. L., & Calzetti, D. 1995, AJ, 110, 1071
- D’Abrusco, R., Fabbiano, G., Djorgovski, G., et al. 2012, ApJ, 755, 92
- D’Abrusco, R., Longo, G., & Walton, N. A. 2009, MNRAS, 396, 223

<sup>19</sup> <https://www.coursera.org/learn/machine-learning>

<sup>20</sup> <http://www.astroml.org/>

- Daniel, S. F., Connolly, A., Schneider, J., Vanderplas, J., & Xiong, L. 2011, *AJ*, 142, 203
- Das, P., & Sanders, J. L. 2019, *MNRAS*, 484, 294
- de Souza, R. S., & Ciardi, B. 2015, *Astronomy and Computing*, 12, 100
- de Souza, R. S., Dantas, M. L. L., Costa-Duarte, M. V., et al. 2017, *MNRAS*, 472, 2808
- Delli Veneri, M., Cavuoti, S., Brescia, M., Longo, G., & Riccio, G. 2019, arXiv e-prints, arXiv:1902.02522
- Dewdney, P. E., Hall, P. J., Schilizzi, R. T., & Lazio, T. J. L. W. 2009, *IEEE Proceedings*, 97, 1482
- D’Isanto, A., Cavuoti, S., Brescia, M., et al. 2016, *MNRAS*, 457, 3119
- D’Isanto, A., Cavuoti, S., Gieseke, F., & Polsterer, K. L. 2018, *A&A*, 616, A97
- D’Isanto, A., & Polsterer, K. L. 2018, *A&A*, 609, A111
- Djorgovski, S. 1995, *ApJL*, 438, L29
- Djorgovski, S. G., Graham, M. J., Donalek, C., et al. 2016, arXiv e-prints: 1601.04385, arXiv:1601.04385
- Donalek, C., Arun Kumar, A., Djorgovski, S. G., et al. 2013, arXiv e-prints, arXiv:1310.1976
- Eatough, R. P., Molkenthin, N., Kramer, M., et al. 2010, *MNRAS*, 407, 2443
- Ellison, S. L., Teimoorinia, H., Rosario, D. J., & Mendel, J. T. 2016, *MNRAS*, 458, L34
- Fadely, R., Hogg, D. W., & Willman, B. 2012, *ApJ*, 760, 15
- Firth, A. E., Lahav, O., & Somerville, R. S. 2003, *MNRAS*, 339, 1195
- Freund, Y., & Schapire, R. E. 1997, *J. Comput. Syst. Sci.*, 55, 119.  
<http://dx.doi.org/10.1006/jcss.1997.1504>
- Fustes, D., Manteiga, M., Dafonte, C., et al. 2013, arXiv e-prints: 1309.2418, arXiv:1309.2418
- Gaia Collaboration, Prusti, T., de Bruijne, J. H. J., et al. 2016, *A&A*, 595, A1
- Galluccio, L., Michel, O., Bendjoya, P., & Slezak, E. 2008, in *American Institute of Physics Conference Series*, Vol. 1082, American Institute of Physics Conference Series, ed. C. A. L. Bailer-Jones, 165–171
- Garcia-Dias, R., Allende Prieto, C., Sánchez Almeida, J., & Ordovás-Pascual, I. 2018, *A&A*, 612, A98
- Gianniotis, N., Kügler, D., Tino, P., Polsterer, K., & Misra, R. 2015, arXiv e-prints, arXiv:1505.00936
- Gianniotis, N., Kügler, S. D., Tiño, P., & Polsterer, K. L. 2016, arXiv e-prints, arXiv:1601.05654
- Giles, D., & Walkowicz, L. 2019, *MNRAS*, 484, 834
- Hartley, P., Flamary, R., Jackson, N., Tagore, A. S., & Metcalf, R. B. 2017, *MNRAS*, 471, 3378
- Hertzsprung, E. 1909, *Astronomische Nachrichten*, 179, 373
- Hocking, A., Geach, J. E., Davey, N., & Sun, Y. 2015, arXiv e-prints: 1507.01589, arXiv:1507.01589
- Hocking, A., Geach, J. E., Sun, Y., & Davey, N. 2018, *MNRAS*, 473, 1108
- Hojnacki, S. M., Kastner, J. H., Micela, G., Feigelson, E. D., & LaLonde, S. M. 2007, *ApJ*, 659, 585
- Huertas-Company, M., Rouan, D., Tasca, L., Soucail, G., & Le Fèvre, O. 2008, *A&A*, 478, 971
- Huertas-Company, M., Primack, J. R., Dekel, A., et al. 2018, *ApJ*, 858, 114
- Hui, J., Aragon, M., Cui, X., & Flegal, J. M. 2018, *MNRAS*, 475, 4494
- Hunter, J. D. 2007, *Computing In Science & Engineering*, 9, 90
- Hyvärinen, A., & Oja, E. 2000, *Neural Netw.*, 13, 411. [http://dx.doi.org/10.1016/S0893-6080\(00\)00026-5](http://dx.doi.org/10.1016/S0893-6080(00)00026-5)
- Ishida, E. E. O., Beck, R., González-Gaitán, S., et al. 2019, *MNRAS*, 483, 2
- Ivezić, Ž., Connolly, A., Vanderplas, J., & Gray, A. 2014, *Statistics, Data Mining and Machine Learning in Astronomy* (Princeton University Press)
- Ivezic, Z., Tyson, J. A., Abel, B., et al. 2008, arXiv: 0805.2366, arXiv:0805.2366
- Jones, E., Oliphant, T., Peterson, P., et al. 2001–, *SciPy: Open source scientific tools for Python*, , [Online; accessed ;today;].  
<http://www.scipy.org/>
- Kaiser, N., Burgett, W., Chambers, K., et al. 2010, in *Proc. SPIE*, Vol. 7733, *Ground-based and Airborne Telescopes III*, 77330E
- Kohonen, T. 1982, *Biological Cybernetics*, 43, 59.  
<https://doi.org/10.1007/BF00337288>

- Kovács, A., & Szapudi, I. 2015, *MNRAS*, 448, 1305
- Krakowski, T., Małek, K., Bilicki, M., et al. 2016, *A&A*, 596, A39
- Krone-Martins, A., Ishida, E. E. O., & de Souza, R. S. 2014, *MNRAS*, 443, L34
- Krone-Martins, A., & Moitinho, A. 2014, *A&A*, 561, A57
- Krone-Martins, A., Delchambre, L., Wertz, O., et al. 2018, *A&A*, 616, L11
- Ksoll, V. F., Gouliermis, D. A., Klessen, R. S., et al. 2018, *MNRAS*, 479, 2389
- Kuncheva, L. I., & Whitaker, C. J. 2003, *Machine Learning*, 51, 181.  
<https://doi.org/10.1023/A:1022859003006>
- Laurino, O., D'Abrusco, R., Longo, G., & Riccio, G. 2011, *MNRAS*, 418, 2165
- Levi, M., Bebek, C., Beers, T., et al. 2013, *ArXiv* : 1308.0847, *arXiv:1308.0847*
- Liu, F. T., Ting, K. M., & Zhou, Z. H. 2008, *IEEE*, 413
- Lochner, M., McEwen, J. D., Peiris, H. V., Lahav, O., & Winter, M. K. 2016, *ArXiv e-prints*: 1603.00882, *arXiv:1603.00882*
- Ma, R., Angryk, R. A., Riley, P., & Filali Boubrahimi, S. 2018a, *ApJS*, 236, 14
- Ma, Z., Xu, H., Zhu, J., et al. 2018b, *arXiv e-prints*, *arXiv:1812.07190*
- MacQueen, J. B. 1967, in *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, ed. L. M. L. Cam & J. Neyman, Vol. 1 (University of California Press), 281–297
- Mahabal, A., Sheth, K., Gieseke, F., et al. 2017, *ArXiv e-prints*, *arXiv:1709.06257*
- Mahabal, A., Djorgovski, S. G., Turmon, M., et al. 2008, *Astronomische Nachrichten*, 329, 288
- Mahabal, A., Rebbapragada, U., Walters, R., et al. 2019, *PASP*, 131, 038002
- Małek, K., Solarz, A., Pollo, A., et al. 2013, *A&A*, 557, A16
- Masci, F. J., Hoffman, D. I., Grillmair, C. J., & Cutri, R. M. 2014, *AJ*, 148, 21
- McInnes, L., Healy, J., & Melville, J. 2018, *arXiv e-prints*, *arXiv:1802.03426*
- Meusinger, H., Brünecke, J., Schalldach, P., & in der Au, A. 2017, *A&A*, 597, A134
- Meusinger, H., Schalldach, P., Scholz, R.-D., et al. 2012, *A&A*, 541, A77
- Miller, A. A. 2015, *ApJ*, 811, 30
- Miller, A. A., Kulkarni, M. K., Cao, Y., et al. 2017, *AJ*, 153, 73
- Möller, A., Ruhlmann-Kleider, V., Leloup, C., et al. 2016, *JCAP*, 12, 008
- Morales-Luis, A. B., Sánchez Almeida, J., Aguerri, J. A. L., & Muñoz-Tuñón, C. 2011, *ApJ*, 743, 77
- Moreno, J., Vogeley, M. S., & Richards, G. 2019, in *American Astronomical Society Meeting Abstracts*, Vol. 233, *American Astronomical Society Meeting Abstracts #233*, 431.02
- Nakoneczny, S., Bilicki, M., Solarz, A., et al. 2018, *arXiv e-prints*, *arXiv:1812.03084*
- Naul, B., Bloom, J. S., Pérez, F., & van der Walt, S. 2018, *Nature Astronomy*, 2, 151
- Norris, R. P. 2017a, *PASA*, 34, e007
- . 2017b, *Nature Astronomy*, 1, 671
- Norris, R. P., Salvato, M., Longo, G., et al. 2019, *arXiv e-prints*, *arXiv:1902.05188*
- Nun, I., Protopapas, P., Sim, B., & Chen, W. 2016, *The Astronomical Journal*, 152, 71. <http://stacks.iop.org/1538-3881/152/i=3/a=71>
- Paatero, P., & Tapper, U. 1994, *Environmetrics*, 5, 111
- Parks, D., Prochaska, J. X., Dong, S., & Cai, Z. 2018, *MNRAS*, 476, 1151
- Pashchenko, I. N., Sokolovsky, K. V., & Gavras, P. 2018, *MNRAS*, 475, 2326
- Pedregosa, F., Varoquaux, G., Gramfort, A., et al. 2011, *Journal of Machine Learning Research*, 12, 2825
- Pérez, F., & Granger, B. E. 2007, *Computing in Science and Engineering*, 9, 21.  
<http://ipython.org>
- Pesenson, M. Z., Pesenson, I. Z., & McCollum, B. 2010, *Advances in Astronomy*, 2010, 350891
- Peth, M. A., Lotz, J. M., Freeman, P. E., et al. 2016, *MNRAS*, 458, 963
- Pichara, K., & Protopapas, P. 2013, *ApJ*, 777, 83
- Pichara, K., Protopapas, P., Kim, D.-W., Marquette, J.-B., & Tisserand, P. 2012, *MNRAS*, 427, 1284
- Plewa, P. M. 2018, *MNRAS*, 476, 3974
- Polsterer, K., Gieseke, F., Igel, C., Doser, B., & Gianniotis, N. 2016, *Parallelized rotation and flipping INvariant Kohonen maps (PINK) on GPUs*, ,
- Protopapas, P., Giammarco, J. M., Faccioli, L., et al. 2006, *MNRAS*, 369, 677
- Qu, M., Shih, F. Y., Jing, J., & Wang, H. 2003, *SoPh*, 217, 157

- Rahmani, S., Teimoorinia, H., & Barmby, P. 2018, *MNRAS*, 478, 4416
- Re Fiorentin, P., Bailer-Jones, C. A. L., Lee, Y. S., et al. 2007, *A&A*, 467, 1373
- Reis, I., Baron, D., & Shahaf, S. 2019, *AJ*, 157, 16
- Reis, I., Poznanski, D., Baron, D., Zasowski, G., & Shahaf, S. 2018a, *MNRAS*, 476, 2117
- Reis, I., Poznanski, D., & Hall, P. B. 2018b, *MNRAS*, 480, 3889
- Richards, J. W., Homrighausen, D., Freeman, P. E., Schafer, C. M., & Poznanski, D. 2012, *MNRAS*, 419, 1121
- Rogers, B., Ferreras, I., Lahav, O., et al. 2007, in *Astronomical Society of the Pacific Conference Series*, Vol. 371, *Statistical Challenges in Modern Astronomy IV*, ed. G. J. Babu & E. D. Feigelson, 431
- Russell, H. N. 1914, *Popular Astronomy*, 22, 275
- Sánchez Almeida, J., Aguerri, J. A. L., Muñoz-Tuñón, C., & de Vicente, A. 2010, *ApJ*, 714, 487
- Sánchez Almeida, J., & Allende Prieto, C. 2013, *ApJ*, 763, 50
- Schawinski, K., Turp, D., & Zhang, C. 2018, in *American Astronomical Society Meeting Abstracts*, Vol. 231, *American Astronomical Society Meeting Abstracts #231*, 309.01
- Schölkopf, B., Williamson, R., Smola, A., Shawe-Taylor, J., & Platt, J. 1999, in *Proceedings of the 12th International Conference on Neural Information Processing Systems, NIPS'99* (Cambridge, MA, USA: MIT Press), 582–588. <http://dl.acm.org/citation.cfm?id=3009657.3009740>
- Segal, G., Parkinson, D., Norris, R. P., & Swan, J. 2018, arXiv e-prints, arXiv:1805.10718
- Shi, T., & Horvath, S. 2006, *Journal of Computational and Graphical Statistics*, 15, 118
- Simpson, J. D., Cottrell, P. L., & Worley, C. C. 2012, *MNRAS*, 427, 1153
- Singh, H. P., Gulati, R. K., & Gupta, R. 1998, *MNRAS*, 295, 312
- Snider, S., Allende Prieto, C., von Hippel, T., et al. 2001, *ApJ*, 562, 528
- Solarz, A., Bilicki, M., Gromadzki, M., et al. 2017, *A&A*, 606, A39
- Storrie-Lombardi, M. C., Lahav, O., Sodre, L., J., & Storrie-Lombardi, L. J. 1992, *MNRAS*, 259, 8P
- Tagliaferri, R., Longo, G., Andreon, S., et al. 2003, *Lecture Notes in Computer Science*, 2859, 226
- Teimoorinia, H., Bluck, A. F. L., & Ellison, S. L. 2016, *MNRAS*, 457, 2086
- van der Maaten, L., & Hinton, G. 2008, *Journal of Machine Learning Research*, 9, 2579. <http://www.jmlr.org/papers/v9/vandermaaten08a.html>
- Vanden Berk, D. E., Shen, J., Yip, C.-W., et al. 2006, *AJ*, 131, 84
- Vanderplas, J., & Connolly, A. 2009, *AJ*, 138, 1365
- Vanderplas, J., Connolly, A., Ivezić, Ž., & Gray, A. 2012, in *Conference on Intelligent Data Understanding (CIDU)*, 47–54
- Vanzella, E., Cristiani, S., Fontana, A., et al. 2004, *A&A*, 423, 761
- Vasconcellos, E. C., de Carvalho, R. R., Gal, R. R., et al. 2011, *AJ*, 141, 189
- Vazdekis, A., Sánchez-Blázquez, P., Falcón-Barroso, J., et al. 2010, *MNRAS*, 404, 1639
- Ward, J. H. 1963, *Journal of the American Statistical Association*, 58, 236. <https://www.tandfonline.com/doi/abs/10.1080/01621459.1963.10500845>
- Weaver, W. B., & Torres-Dodgen, A. V. 1995, *ApJ*, 446, 300
- Wright, D. E., Smartt, S. J., Smith, K. W., et al. 2015, *MNRAS*, 449, 451
- Yang, T., & Li, X. 2015, *MNRAS*, 452, 158
- Yong, S. Y., King, A. L., Webster, R. L., et al. 2018, ArXiv e-prints, arXiv:1806.07090
- York, D. G., Adelman, J., Anderson, Jr., J. E., et al. 2000, *AJ*, 120, 1579
- Zhang, J.-n., Wu, F.-c., Luo, A.-l., & Zhao, Y.-h. 2006, *Chinese Astronomy and Astrophysics*, 30, 176
- Zucker, S., & Giryes, R. 2018, *AJ*, 155, 147