Light curve classification using distance metrics

Siddharth Chaini*†, Ashish Mahabal‡§, Federica Bianco†, Ajit Kembhavi¶, and Sukanta Panda*
*Department of Physics, Indian Institute of Science Education and Research, Bhopal 462066, India
†Department of Physics and Astronomy, University of Delaware, Newark, DE 19716-2570, USA
‡Division of Physics, Mathematics and Astronomy, California Institute of Technology, Pasadena, CA 91125, USA
§Center for Data Driven Discovery, California Institute of Technology, Pasadena, CA 91125, USA
¶Inter University Centre for Astronomy and Astrophysics (IUCAA), Pune 411007, India

***Abstract*—The rise of extensive sky surveys has ushered in an era of big data in time-domain astronomy, which has made data science and machine learning an essential tool for classification. In this paper, we present the use of distance metrics for classification of light curves. In particular, we classify light curves of variable stars by comparing the distances between objects of different classes. A distance is an abstract quantity that tells us about the closeness of two objects, and a distance metric is a rule to calculate this distance. We demonstrate the use of various distance metrics for the dimensionality reduction and classification of light curves and compare the performance of different distance metrics for different classes. We find that certain classes and features are easier to separate with specific metrics. The use of different distance metrics for classification and dimensionality reduction is an approach that has not been explored in time-domain astronomy yet.**

***Index Terms*—some, keywords**

I. Introduction

Over the last few decades, time-domain astrophysics has experienced rapid growth. The significant factors behind this growth have been the advent of large-scale sky surveys like the Sloan Digital Sky Survey (SDSS; [1]), the Catalina Real-Time Transient Survey (CRTS; [2]) and the Zwicky Transient Facility (ZTF; [3]) (See [4] for more surveys.) accompanied by advances in computer data storage. By observing how millions of astronomical objects change over time, we can look at changes in the night sky that were once impossible to detect. The Rubin Observatory Legacy Survey of Space and Time (LSST; [5]), is expected to observe over 37 billion objects over its 10-year lifespan. However, this opportunity for new discoveries in the data deluge is accompanied by a data-intensive challenge. Manual human classification of all objects has become impossible, and we need to deploy methods that can automate the classification and identification of objects of interest. Machine learning is one such approach to it.

Machine learning refers to a class of computer algorithms in which the computer automatically learns patterns from data without being programmed. Thus, in the context of the big datasets and the need for automation, machine learning has become essential in modern astrophysics. One such machine learning task which is critical to time-domain astrophysics is the classification of astronomical objects based on how their brightness changes with time . Machine learning for this involves extracting features from the light curves, which are then fed to the machine learning algorithm. However, as the number of features used in a machine learning model increases, prediction becomes difficult and computationally expensive. This complexity is mainly because of the "curse of dimensionality" [6], which leads to a decrease in performance and an increase in computational complexity. Furthermore, this is accompanied by data abstraction, which results in the model behaving like a black-box, and makes explainability difficult.

To help tackle the above problems, we can reduce dimensionality by transforming our data. This involves mapping our data from a high dimensional space into a low dimensional space while ensuring that the low dimensional representation retains the properties of the original data.

A distance is a scalar quantity that tells us how far away two objects are, while a distance metric refers to the method by which this distance is calculated. Because the distance between objects from the same class is smaller than objects from different classes, an analysis of these distances can be used to separate and classify light curves.

In this paper, we compare the use of different distance metrics to classify light curves and reduce the dimensionality of their feature space. We look at 18 different metrics and compare their performance for different types of classification problems for variable stars. In addition, we also find the most important features associated with a particular distance metric and reduce the dimensionality of the feature space to these top features. The use of different distance metrics in this goal is an approach that has not been explored in time-domain astrophysics yet.

The paper is organised as follows:

II. Distances in Machine Learning

The notion of distance is intuitive to most of us. It tells us about the degree of closeness of two physical objects or ideas. The shorter the distance, the closer the objects or ideas are. However, the existence of such a quantity does not tell us how to calculate it. Because of this, we can have different types of distances, each calculated differently. These distances can have different physical meanings, but they can also be entirely abstract. A formal definition has been listed in **??**

For most machine learning tasks, we cannot feed the data directly to the machine learning model. For this reason, we generally extract *features* from the training data and transform it into a new space called the feature space. The dimensionality of the feature space is the number of features (columns of the data) we extract from the training data, and the feature space

Corresponding author: S. Chaini (sidchaini@gmail.com).

forms the metric space. The points in this space are given by the individual data points (rows of the data). The feature space is similar to the n-dimensional real coordinate space, $\mathbb{R}^n$.

Now, we can define a variety of distance metrics in this higher dimensional feature space, as per definition A.1. Because the distance is always a positive real number, it can be compared easily for different points even if the feature space is highly dimensional.

Distance metrics see their use in a variety of machine learning algorithms, in both supervised (for e.g., k-Nearest Neighbours; [7]) and unsupervised settings (for e.g., kernel density estimation; [8]). We have listed each distance metric that we use in B.

III. Data

A. Catalog and Raw Light Curves

To develop our methodology for light curve classification with distance metrics, we choose to work only with variable stars. However, because the only input to our model consists of light curves, we note that our methodology is transferable to light curves of all variables as well as transient classes.

Our raw dataset consists of light curves from the Zwicky Transient Facility (ZTF). To select variable stars, we use the classifications from the catalog provided by [9], which is based on the ZTF Data Release 2 (DR2). The catalog consists of a total of 781,602 identified variable stars. However, the number of objects in each class is imbalanced, with the catalog containing 369,707 EW variables, but just 1,262 Cepheid variables. This forms an imbalanced dataset, as each class has an unequal number of objects. One of the issues with imbalanced classification problems is that models trained on an imbalanced dataset have a hidden bias due to the relative frequencies when the machine learns from the data and perform poorly on minority classes [10]. Thus, we choose only 1,000 objects of each class from [9]'s dataset to work with a balanced number of classes throughout this thesis.

Our raw dataset thus consists of 100,000 light curves (10,000 for each of the 10 classes listed in Tab. **??**). We download these from the catalog website[1].

B. Feature Extraction

Since the ZTF is a ground-based telescope, the light curves obtained are sparse and unevenly sampled, noisy and heteroskedastic. On top of that, even objects belonging to the same classes have different noise levels and different magnitudes based on their distance from the earth. Because of this, the direct comparison of light curves is difficult. However, we can instead extract different features from these light curves - attributes that give us some information about the light curve. These features allow us to compare different light curves better and are fed to the machine learning algorithm for classification.

To extract features from the light curves, we use the `lc_classifier` module [11] on Python. All of `lc_classifier`'s features are based on the light curves only. Most (50) features are calculated separately for the g and r passband, while some (8) features are calculated using both the g and r observations. In addition to this, we also use 5 features for each passband provided by [9]. This gives us a total of 118 features for every light curve in our dataset.

Most features are based on the light curve statistics (e.g. `Amplitude` - based on the difference of highest and lowest magnitudes), while a few are based on parameters obtained after fitting a model (e.g. `Multiband_Period` - the period is obtained by fitting a periodogram). A list of all the features, along with their descriptions, is available in Section 3.1 and Appendix A of [12].

C. Data Cleaning

To clean our dataset, we first start by removing those objects from our dataset for which feature extraction with `lc_classifier` failed - thus removing all objects for which even a single feature was not calculated. For most classes, the number of such objects are 3-8%, but we find that the classes Mira, CEP and SR have a higher fit failure rate, thus leaving us with only 640 Miras, 713 CEPs and 741 SRs . However, to keep the dataset balanced, we randomly drop values from our dataset such that each class has exactly 600 objects (600 because it as it is the highest round number less than 640).

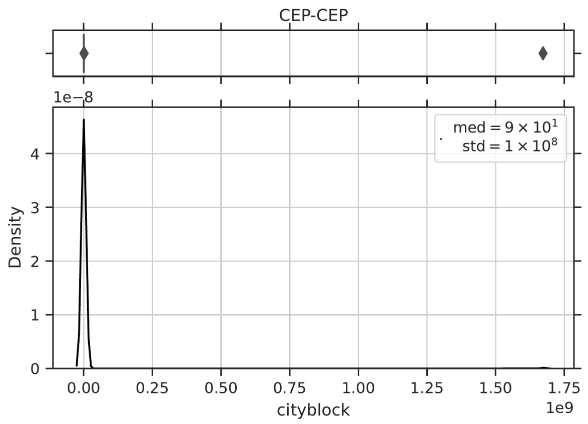
Secondly, since [9]'s catalog was obtained through spectroscopic measurements, we hypothesise that there might be a few outliers in the dataset - objects that might have been mislabelled.

To explore if this is indeed the case, we first calculate all the pair-wise distances among all classes with the Cityblock metric (8) and then plot their distribution. We visualise this distribution of the distances through a box plot and a density plot, both of which help show us the spread of our distribution. This has been illustrated for Cepheid variables in Fig. 1a.

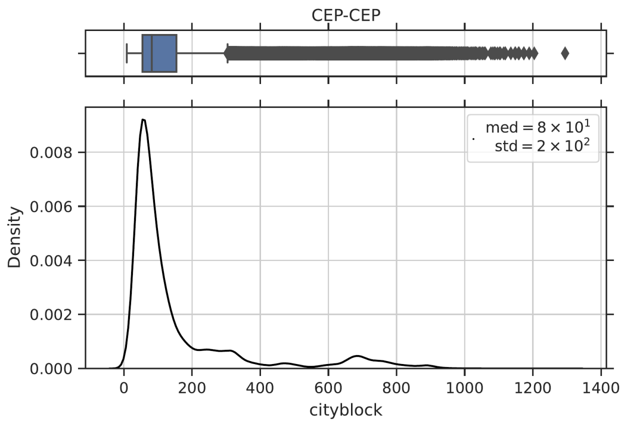
We see a prominent peak centred near 0, accompanied by a small peak at a vast distance. Because the standard deviation of this distribution is a couple of magnitudes higher than the median, we note that these objects are probably misclassifications in [9]'s catalog or are objects for which feature extraction did not work well. We thus claim that the objects responsible for this second peak are outliers, i.e. do not represent most objects, and remove them from our dataset. This is done by dropping the objects responsible for the top 5% of the distances. In addition, we also drop the bottom 2% of the objects responsible for the low distances, leaving us with 558 objects per class. We do this to remove any possible duplicates or re-observations. After removing these outliers, our dataset is much better distributed without significant disruptions. The histogram for Cepheid variables after outlier removal is illustrated in Fig. 1b.

Finally, we are left with a 'clean' dataset of 558 objects of each class - CEP, RR, RRc, DSCT, Mira, SR, RSCVN, BYDra, EA, EW.

[1]http://variables.cn:88



(a) Before outlier removal.



(b) After outlier removal

Fig. 1: Density plot for Cityblock distance distribution between Cepheid variables

IV. Classification

A. Classification Algorithm

We use a custom algorithm called `LCDistanceClassifier` to classify the light curves, which is motivated by k-Nearest Neighbours. We implement this on Python using scikit-learn's API [13]. We describe our algorithm's training and testing procedure below.

1) Training: For all objects in the training set, we calculate the median and the standard deviation per feature per class. The median set represents what an average class's features looks like, and we use it as a representative for each class.

An example of the median set has been displayed in Tab. I.

Class	`Period_band_r`	`Amplitude_r`	`R21_r`	`Skew_r`
CEP	3.86	0.25	0.27	-0.16
RR	0.56	0.35	0.45	-0.52
RRc	0.33	0.20	0.15	0.01
DSCT	0.09	0.08	0.17	-0.14

TABLE I: An example of the median set for 4 r-band features and 4 classes.

2) Predicting: To classify a test object, we first choose a distance metric, and then calculate the distance between the test object and the median set for each class from the training step. When calculating the distance, we also scale it by the inverse of the standard deviation for every feature. This step thus tells us how many standard deviations away the test object is from the median set (for a particular metric). This scaling can be denoted by:

$$d_\sigma(u,v) = \sum_i \frac{1}{\sigma_i} d_i(u_i, v_i), \tag{1}$$

where i denotes the dimension, d_i denotes the ith component of the distance metric d in the ith dimension and σ_i denotes the standard deviation in the ith dimension.

Finally, once all the distances have been calculated, we look at the class for which the distance is minimum. This is chosen as the predicted class for a test object. This is then repeated for all test objects to obtain the final class predictions.

3) Scoring:

a) F_1 Score: Once we have made the predictions for all objects in the test set, we measure its performance by calculating the F_1 score. The F_1 score is given by

$$F_1 = \frac{\text{TP}}{\text{TP} + \frac{1}{2}(\text{FP} + \text{FN})}, \tag{2}$$

where TP is the number of true positives, FP is the number of false positives, and FN is the number of false negatives. The higher the F_1 score, the better the classification, with 100% being the best possible score (all predictions correct) and 0% being the worst possible score (all predictions wrong). For more than two classes, we take the average (*macro* mode) of the F_1 scores for each class. Because we consistently maintain class balance, we do not require any weights when calculating the F_1 score.

We note that, instead of using a separate training and testing set, we use the cross-validation technique [14]. In this, we split our dataset into 5 parts or 5-folds. We choose four folds as our training set and one fold as our testing set. This is repeated such that all folds are chosen to be the test set exactly once. Finally, we calculate the F_1 score for all 5-folds and average the scores to obtain the final score.

b) Confidence Parameter (c_p): In addition to the F_1 score, we also define a confidence parameter to tell us how confident the prediction by `LCDistanceClassifier` is.

The confidence parameter, c_p, for a prediction is given by

$$c_p = \frac{|d_{\text{winner}} - d_{\text{runner-up}}|}{d_{\text{winner}}}, \tag{3}$$

where d_{winner} is the distance to the class to which the distance is minimum (the winner is the class that was chosen as the prediction), and $d_{\text{runner-up}}$ is the distance to the class to which the distance is second minimum (the runner-up is the class that would have been chosen had the winner not been present). The confidence parameter thus tells us the relative difference between the distances of the winner and the runner-up. If this difference is large, we then say that the classifier chose the winning class with high confidence and vice versa for low confidence.

For example, if the distance to the winner is 10 and the distance to the runner up is 100, then the confidence $c_p = \frac{100-10}{10} = 9$. On the other hand, if the distance to the winner is 99 and the distance to the runner up is 100, then the confidence $c_p = \frac{100-99}{99} = 0.01$.

The motivation behind having the confidence values is that they serve as an additional measure of the quality of the classification. A high F_1 score implies that the predictions made by our classifier are correct and agree with [9]'s catalog. In contrast, high confidence tells us that the classifier is confident about its prediction. Ideally, a perfect classifier would have a F_1 score of 100% with a high confidence value for each prediction. Thus, by choosing the classifier with the top F_1 score followed by the highest confidence value, we can determine which distance metric is the best for a given classification problem.

V. Feature Selection and Dimensionality Reduction

To reduce the dimensionality of the feature space, we employ the Forward Sequential Feature Selection strategy [15] with `LCDistanceClassifier`. We start by choosing the single feature (from the 118) that maximises our 5-fold cross-validation F_1 score with just that feature. Once we have selected this, we then choose the next feature, which, when added to the feature set, maximises the 5-fold cross-validation F_1. We repeat this process until we have selected all of the 118 features. The total number of combinations are thus,

$$\begin{aligned}\text{No. of combinations} &= \sum_{i=0}^{n} \binom{n-i}{1} \\ &= \frac{n(n+1)}{2} \qquad (4)\\ &= \mathcal{O}(n^2) \qquad (5)\end{aligned}$$

Here, because $n = 118$, we perform a total of 7021 calculations for each metric for a classification problem.

Thus, this gives us the order of importance of the features for a particular classification problem - the feature selected first has the highest importance, while the feature selected last has the least importance. In addition, the Sequential Feature Selection strategy also allows us to find out how the performance changes with the number of features included. Finally, once we know the small set of most important features, we can drop the remaining unimportant features to reduce the dimensionality of our feature space.

Because each metric and class behaves differently, we expect the top features to depend on both the classification problem, as well as the metric used. We present our results for this in detail in the next section VI.

VI. Results

Acknowledgment

References

[1] D. G. York, J. Adelman, J. Anderson, John E. *et al.*, "The Sloan Digital Sky Survey: Technical Summary," *AJ*, vol. 120, no. 3, pp. 1579–1587, Sep. 2000.

[2] S. G. Djorgovski, A. J. Drake, A. A. Mahabal *et al.*, "The Catalina Real-Time Transient Survey (CRTS)," *arXiv:1102.5004 [astro-ph]*, Feb. 2011.

[3] E. C. Bellm, S. R. Kulkarni, M. J. Graham *et al.*, "The Zwicky Transient Facility: System Overview, Performance, and First Results," *Publications of the Astronomical Society of the Pacific*, vol. 131, no. 995, p. 018002, Jan. 2019.

[4] S. G. Djorgovski, A. Mahabal, A. Drake *et al.*, *Sky Surveys*. Dordrecht: Springer Netherlands, 2013, pp. 223–281. [Online]. Available: https://doi.org/10.1007/978-94-007-5618-2_5

[5] Ž. Ivezić, S. M. Kahn, J. A. Tyson *et al.*, "LSST: From Science Drivers to Reference Design and Anticipated Data Products," *The Astrophysical Journal*, vol. 873, no. 2, p. 111, Mar. 2019.

[6] C. M. Bishop, *Pattern Recognition and Machine Learning*, ser. Information Science and Statistics. New York: Springer, 2006.

[7] H. A. Abu Alfeilat, A. B. Hassanat, O. Lasassmeh *et al.*, "Effects of Distance Measure Choice on K-Nearest Neighbor Classifier Performance: A Review," *Big Data*, vol. 7, no. 4, pp. 221–248, Dec. 2019.

[8] Y. He, W. Chen, Y. Chen, and Y. Mao, "Kernel Density Metric Learning," in *2013 IEEE 13th International Conference on Data Mining*. Dallas, TX, USA: IEEE, Dec. 2013, pp. 271–280.

[9] X. Chen, S. Wang, L. Deng *et al.*, "The Zwicky Transient Facility Catalog of Periodic Variable Stars," *The Astrophysical Journal Supplement Series*, vol. 249, no. 1, p. 18, Jul. 2020.

[10] B. Krawczyk, "Learning from imbalanced data: Open challenges and future directions," *Progress in Artificial Intelligence*, vol. 5, no. 4, pp. 221–232, Nov. 2016.

[11] I. R. Jainaga, J. Arredondo, C. Valenzuela *et al.*, "Alercebroker/lc_classifier: Release 1.2.3-P," Zenodo, Aug. 2021.

[12] P. Sánchez-Sáez, I. Reyes, C. Valenzuela *et al.*, "Alert Classification for the ALeRCE Broker System: The Light Curve Classifier," *The Astronomical Journal*, vol. 161, no. 3, p. 141, Mar. 2021.

[13] L. Buitinck, G. Louppe, M. Blondel *et al.*, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

[14] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI'95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, p. 1137–1143.

[15] F. Ferri, P. Pudil, M. Hatef, and J. Kittler, "Comparative study of techniques for large-scale feature selection* *This work was suported by a SERC grant GR/E 97549. The first author was also supported by a FPI grant from the Spanish MEC, PF92 73546684," in *Machine Intelligence and Pattern Recognition*. Elsevier, 1994, vol. 16, pp. 403–413.

[16] M. M. Deza and E. Deza, *Encyclopedia of Distances*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

Appendix

A. Distance Metric Definitions

Definition A.1: The distance d between two points, in a set X, is a function $d : X \times X \rightarrow [0, \infty)$ that gives a distance between each pair of points in that set such that, for all $x, y, z \in X$, the following properties hold:

1) $d(x, y) = 0 \iff x = y$ (identity of indiscernibles)
2) $d(x, y) = d(y, x)$ (symmetry)
3) $d(x, y) \leq d(x, z) + d(z, y)$ (triangle inequality)

Corollary: From the above three axioms, the following condition also holds:

$$d(x, y) \geq 0, \text{for all } x, y \in X \tag{6}$$

Example: The Euclidean distance between two points in $\mathbb{R}^2$ is an example of a distance. Similarly, we can define a distance between any set of points such that the definition A.1 is satisfied.

Definition A.2: A metric space (X, d) is a set X equipped with a metric d.

Example: The 2-dimensional $\mathbb{R}^2$ plane with the Euclidean distance is an example of a metric space. Similarly, we can also define metrics on matrices, functions, sets of points or any other mathematical object as long as it satisfies the definition A.2.

Now, we note the difference between the terms *distance* and *distance metric* (often shortened to just *metric*).

Definition A.3: The term *metric* refers to the way of computing the distance between any two elements of the set. Meanwhile, the term *distance* refers to the scalar value obtained after using a particular metric on a pair of elements in the set.

Example: The Euclidean metric for $\mathbb{R}^2$ is given by $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$, while the Euclidean distance between the points $(0, 0)$ and $(1, 1)$ is given by $d = \sqrt{2}$.

B. List of Metrics

We used the following metrics for all distance calculations in this paper. For a detailed description of each metric, see [16].

1) **Euclidean metric**:

$$d(u, v) = \sqrt{\sum (v_i - u_i)^2} \tag{7}$$

The Euclidean metric is the most widely used metric, and represents the length of the shortest straight line joining 2 points.

2) **Cityblock metric**:

$$d(u, v) = \sum |v_i - u_i| \tag{8}$$

The Cityblock metric gives the shortest path between 2 points if we restrict movement to the grid. Named after the grid layout of cities, this is the shortest distance for a taxi to traverse between two places. It is also known as the Manhattan metric.

3) **Bray-Curtis metric**:

$$d(u, v) = \frac{\sum |u_i - v_i|}{\sum |u_i + v_i|} \tag{9}$$

The Bray-Curtis metric is a modified version of the Cityblock metric. Note that the summation is performed independently on the numerator and denominator before division.

4) **Canberra metric**:

$$d(u, v) = \sum_i \frac{|u_i - v_i|}{|u_i| + |v_i|} \tag{10}$$

The Canberra metric is another modified version of the Cityblock metric, whose denominator is slightly different from the Bray-Curtis metric.

5) **Chebyshev metric**:

$$d(u, v) = \max \left(|u_i - v_i|\right) \tag{11}$$

The Chebyshev metric is also known as the chessboard metric. It represents the minimum number of moves needed by the King chess piece to move from one point to another.

6) **Correlation metric**:

$$d(u, v) = 1 - \frac{(u - \bar{u}) \cdot (v - \bar{v})}{\|(u - \bar{u})\|_2 \|(v - \bar{v})\|_2} \tag{12}$$

The Correlation metric is based on the Pearson correlation coefficient between 2 points, it is given by:

7) **Cosine metric**:

$$d(u, v) = \frac{\sum (u_i - \bar{u}) \cdot (v_i - \bar{v}))}{\sqrt{\sum (u_i - \bar{u}) \cdot (u_i - \bar{u}))} \sqrt{\sum (v_i - \bar{v}) \cdot (v_i - \bar{v}))}} \tag{13}$$

The Cosine metric is based on the cosine similarity, which depends only on the angle between two vectors and not their magnitude.

8) **Soergel metric**:

$$d(u, v) = \frac{\sum |u_i - v_i|}{\sum \max \{u_i, v_i\}} \tag{14}$$

The Soergel metric is another modified version of the Cityblock metric but only has a $\max$ term in the denominator. Note that the summation is performed independently on the numerator and denominator before division.

9) **Kulczynski metric**:

$$d(u, v) = \frac{\sum |u_i - v_i|}{\sum \min \{u_i, v_i\}} \tag{15}$$

The Kulczynski metric is similar to the Soergel metric, with the difference that its denominator consists of a minimum instead of a maximum.

10) **Wave–Edges metric**:

$$d(u, v) = \sum \frac{|u_i - v_i|}{\max \{u_i, v_i\}} \tag{16}$$

The Wave-Edges metric is a slight modification of the Soergel metric, where the summation is done after division.

11) **Jaccard metric**:

$$d(u, v) = \frac{\left(\sum u_i - v_i\right)^2}{\sum u_i^2 + \sum v_i^2 - \sum u_i v_i} \tag{17}$$

The Jaccard metric is based on the Jaccard similarity coefficient, which is a measure of similarity between 2 sets.

12) **Maryland Bridge metric**:

$$d(u,v) = 1 - \frac{1}{2}\left(\frac{\sum u_i v_i}{\sum u_i^2} + \frac{\sum u_i v_i}{\sum v_i^2}\right) \tag{18}$$

The Maryland-Bridge metric is based on the Maryland Bridge similarity.

13) **Motyka metric**:

$$d(u,v) = \frac{\sum \max\{u_i, v_i\}}{\sum (u_i + v_i)} \tag{19}$$

Unlike the previous metrics, the Motyka metric has a maximum term in the numerator and a summation term in the denominator.

14) **Lorentzian metric**:

$$d(u,v) = \sum \ln(1 + |u_i - v_i|) \tag{20}$$

Because the Lorentzian metric is given by a natural log, it is more sensitive to changes in the input vectors, as the log blows up quickly.

15) **Clark metric**:

$$d(u,v) = \left(\frac{1}{n}\sum\left(\frac{u_i - v_i}{|u_i| + |v_i|}\right)^2\right)^{\frac{1}{2}} \tag{21}$$

The Clark metric can be thought of as a combination of the Euclidean metric and the Canberra metric, where the ratio is similar to the Canberra metric, which is squared, summed and taken a square root of, like the Euclidean metric.

16) **Meehl metric**:

$$d(u,v) = \sum_{1 \le i \le n-1} (u_i - v_i - u_{i+1} + v_{i+1})^2 \tag{22}$$

The Meehl metric mixes terms from consecutive dimensions, thus making the ordering of the dimensions significant.

17) **Hellinger metric**:

$$d(u,v) = \sqrt{2\sum\left(\sqrt{\frac{u_i}{\bar{u}}} - \sqrt{\frac{v_i}{\bar{v}}}\right)^2} \tag{23}$$

The Hellinger metric is a metric that is commonly used to compare two probability distributions but can also be used with numeric data.

18) **Symmetric χ^2 metric**:

$$d(u,v) = \sqrt{\sum \frac{\bar{u} + \bar{v}}{n(\bar{u} \cdot \bar{v})^2} \cdot \frac{(u_i \bar{v} - v_i \bar{u})^2}{u_i + v_i}} \tag{24}$$

The symmetric χ^2 metric is a kind of a weighted Euclidean metric, where terms between the two vectors are mixed.