Machine learning Calabi-Yau hypersurfaces

David S. Berman,[1,*] Yang-Hui He,[2,3,4,5,†] and Edward Hirst[3,2,‡]

[1]*Centre for Theoretical Physics, School of Physics and Astronomy, Queen Mary University of London,
327 Mile End Road, London E1 4NS, United Kingdom*
[2]*London Institute for Mathematical Sciences, Royal Institution, London W1S 4BS, United Kingdom*
[3]*Department of Mathematics, City, University of London, EC1V 0HB, United Kingdom*
[4]*Merton College, University of Oxford, OX1 4JD, United Kingdom*
[5]*School of Physics, NanKai University, Tianjin, 300071, People's Republic of China*

(Received 20 January 2022; accepted 23 February 2022; published 9 March 2022)

We revisit the classic database of weighted-$\mathbb{P}^4$s which admit Calabi-Yau 3-fold hypersurfaces equipped with a diverse set of tools from the machine-learning toolbox. Unsupervised techniques identify an unanticipated almost linear dependence of the topological data on the weights. This then allows us to identify a previously unnoticed clustering in the Calabi-Yau data. Supervised techniques are successful in predicting the topological parameters of the hypersurface from its weights with an accuracy of $R^2 > 95\%$. Supervised learning also allows us to identify weighted-$\mathbb{P}^4$s which admit Calabi-Yau hypersurfaces to 100% accuracy by making use of partitioning supported by the clustering behavior.

DOI: 10.1103/PhysRevD.105.066002

I. INTRODUCTION

Artificial intelligence has now permeated through all disciplines of human enterprise. Machine-learning (ML) has become, in this age driven by big data, as indispensable a tool as calculus was to the Age of Enlightenment [1]. Perhaps surprisingly, noiseless, pure mathematical data can often be learned at high precision, indicating underlying formulas which have not yet been uncovered from traditional analyses. Examples of this data driven approach to mathematics may be seen in applications of ML to: the string theory landscape [2–6]; abstract algebra [7]; modern number theory [8,9]; and graph theory [10]. It is hoped that ML might reveal structures in the very nature of mathematics [11] and mathematical intuition [12], deeply embedded in the *mathematical data*. Apart from ML itself, the tools that have been developed to enable ML have provided significant new capabilities for old problems. This is exemplified by the use of the autodifferentiation capabilities of `Tensorflow` to explore the possible vacua of various gauged supergravities, see for example [13–18].

Among its various virtues, string theory pioneered the data-mining of such mathematical data. One should be mindful that this was done shortly after the beginnings of string phenomenology in the late 1980s, long before the dawn of the modern era of "big data" and modern readily available ultrafast computing power. Indeed, when Calabi-Yau manifolds were realized to be [19] the standard solution to vacuum configurations (see [20] for a brief, and [21] a longer, pedagogical review), and hence low-energy particle physics, a program was introduced by the physics community to compile one of the first databases in algebraic geometry.

These were some of the earliest appearances of "big data" in mathematics, beyond compiling digits of π or large primes. The first dataset was the so-called CICYs, which stands for "complete intersection Calabi-Yau manifolds" in products of complex projective spaces [22,23]; which can be thought of as a generalization of the famous quintic 3-fold in $\mathbb{P}^4$. Appropriately, one of the first ML experiments in geometry was performed on this dataset [2]. Over the last few years, the initial success has been vastly improved by using more and more sophisticated neural network (NN) architectures and machine learning techniques [24–44].

Yet, the CICY dataset has a peculiarity: it is skewed toward negative Euler number. This would have occurred to Candelas *et al.* since they knew about mirror symmetry. Since the exchange of the two Hodge numbers $(h^{1,1}, h^{2,1})$ would reverse the sign of the Euler number χ; the conjecture that to every Calabi-Yau 3-fold with $(h^{1,1}, h^{2,1})$ there is a mirror with these exchanged would imply the negation of χ.

[*] d.s.berman@qmul.ac.uk
[†] hey@maths.ox.ac.uk
[‡] Corresponding author.
edward.hirst@city.ac.uk

Therefore, as the second database of geometries in string theory, another generalization of the quintic was undertaken by placing *weights* on the ambient $\mathbb{P}^4$ and considering a single, generic Calabi-Yau hypersurface therein [45]. This produced a much more balanced set of Calabi-Yau 3-folds with ± Euler numbers, and the rough outline of the famous "mirror plot" of the distributions of $2(h^{1,1} - h^{2,1})$ vs $(h^{1,1} + h^{2,1})$ could already be seen to emerge.

All these datasets were subsequently subsumed into the dataset created through the extraordinary work of Kreuzer and Skarke [46,47]. This set contains the Calabi-Yau hypersurfaces in toric varieties. (Since weighted projective spaces are special types of toric varieties, the set described in [45] is a subset of the Kreuzer-Sharke set.)

However, the Kreuzer-Sharke dataset is of astronomical size, containing some half-billion members. While ML of this set is in progress [48], the much more manageable list of hypersurfaces in weighted $\mathbb{P}^4$, numbering around 8000 (comparable to the CICYs) is natural choice of geometries to study and apply the latest methods from data science.

Thus our motivation is clear. We shall revisit the classic database of [45] with a modern perspective, continuing the paradigm of machine-learning the string theory landscape and the resulting emergent mathematical structures, using tools from the `sci-kit learn` library [49] implemented in PYTHON. The paper is organized as follows. In Sec. II we begin with a rapid review of the mathematics of our Calabi-Yau hypersurfaces, emphasizing on the data structure. In Sec. III, we proceed with analyses of the data, using methods which were unavailable at the time of their creation, such as principle component analysis and topological data analysis. We then use neural-networks to machine learn the dataset in Sec. IV. We conclude with a summary and outlook in Sec. V.

II. CHARACTERIZING THE CALABI-YAU HYPERSURFACES

The dataset of focus in this study is that of weighted projective spaces $\mathbb{P}^4_{\mathbb{C}}(w_i)$, which admit Calabi-Yau (CY) three-fold hypersurfaces within them.

This dataset was constructed in the early 1990s alongside other efforts to expand the CY landscape for use in Landau-Ginzburg models and superstring compactification [45,46,50–52].

A generic weighted projective space generalizes the concept of a projective space, defined by taking some $\mathbb{C}^{n+1}$ with coordinates $\{z_1, z_2, \ldots, z_{n+1}\}$ and performing an identification with weights w_i such that

$$(z_1, z_2, \ldots, z_{n+1}) \sim (\lambda^{w_1} z_1, \lambda^{w_2} z_2, \ldots, \lambda^{w_{n+1}} z_{n+1}), \qquad (2.1)$$

$\forall\, \lambda \in \mathbb{C}$, hence defining the projective space $\mathbb{P}^n$ with these $n + 1$ homogeneous coordinates. For the projective spaces in consideration n takes the value of 4, and the space is hence defined with 5 weights. These weights are coprime as a set, such that the projective space definition is free from redundancy from different weight combinations.

Within these weighted projective spaces one can embed hypersurfaces defined by polynomials in these homogeneous coordinates. Of particular interest to physicists are those hypersurfaces which are CY in nature. A defining property of CY manifolds is the vanishing of the first Chern class, and for this to hold within the projective space the hypersurface's polynomial has to have degree $d = \sum_i (w_i)$.

It should be noted that the identifications that are used in constructing the projective space lead to singular sets, which the hypersurfaces can intersect with suitable resolution. To be consistently defined over these singular sets another property of the polynomial is required: *transversity*. The transversity property implies that the polynomial equation and its derivative share no common solutions, and this condition translates into a condition on the projective space weights:

$$\forall\, w_i \,\exists\, w_j s.t. \frac{\sum_k (w_k) - w_j}{w_i} \in \mathbb{Z}^+. \qquad (2.2)$$

However as exemplified in [45], this condition is necessary but not sufficient for the surface to be CY. It is therefore of interest to consider the extent to which each of these 5-vector weights properties contribute to determine the very special CY property; and it is this question we look to probe with new tools from data analysis, and machine-learning.

It has been shown that only a finite number of possible weights permit these CY hypersurfaces. In fact, the dataset of weights consists of just 7555 5-vectors of transverse coprime integers.

Beyond learning the CY property explicitly, we are also interested in exploring if the topological features of the Calabi-Yau can be learnt from the weights. Of specific importance are the nontrivial Hodge numbers, $h^{1,1}$ and $h^{2,1}$, which describe the manifolds cohomology, and the Euler number, χ. These all have a variety of uses in determining physical phenomena. The formula for Hodge numbers comes from expansion of the Poincaré polynomial $Q(u, v) \coloneqq \sum_{p,q} h^{p,q} u^p v^q$, the generating function of the Hodge numbers $h^{p,q}$; while the formula for Euler number has a direct form [50,53–55]. Specifically these are

$$Q(u,v) = \frac{1}{uv} \sum_{l=0}^{\sum_i (w_i)} \left[\prod_{\tilde{\theta}_i(l) \in \mathbb{Z}} \frac{(uv)^{q_i} - uv}{1 - (uv)^{q_i}} \right]_{\text{int}} \left(v^{\text{size}(l)} \left(\frac{u}{v}\right)^{\text{age}(l)} \right),$$

$$\chi = \frac{1}{\sum_i (w_i)} \sum_{l,r=0}^{\sum_i (w_i) - 1} \left[\prod_{i | l q_i \& r q_i \in \mathbb{Z}} \left(1 - \frac{1}{q_i}\right) \right], \qquad (2.3)$$

for weights w_i, normalized weights $q_i = w_i / \sum_i (w_i)$, and u, v are the dummy variables of the Poincaré polynomial.

For $Q(u,v)$, $\tilde{\theta}_i(l)$ is the canonical representative of lq_i in $(\mathbb{R}/\mathbb{Z})^5$, $\mathrm{age}(l) = \sum_{i=0}^{4} \tilde{\theta}_i(l)$ and $\mathrm{size}(l) = \mathrm{age}(l) + \mathrm{age}(\sum_i(w_i) - l)$. Note also for χ, where $\forall\, i lq_i \,\mathrm{or}\, rq_i \notin \mathbb{Z}$ then the product takes value 1 [55].

Both formulas require significant computation, involving many nontrivial steps. Even if we realize this dataset in the language of the toric geometry of [46,56], the formulas involve nontrivial sums over faces of various dimension. It is consequently interesting to examine the performance of machine-learning methods in learning the Euler number/Hodge numbers from the weights, and perhaps predicting the results through the use of possible hidden structures which we hope to uncover in the weights and weight distributions.

III. DATA ANALYSIS

Before we apply the supervised machine-learning methods described in Sec. IV, let us provide some analysis of the fundamentals of the dataset through the use of *principal component analysis* (PCA), *topological data analysis* (TDA), and other unsupervised machine-learning methods.

A. Datasets

In addition to the CY dataset which forms the central focus of this study, we will construct some auxiliary datasets that will help in assessing the learning of the Calabi-Yau property. These are equivalent datasets of 5-vectors that possess fewer of the necessary properties required to meet the Calabi-Yau property.

The 4 datasets (including the original CY dataset) are composed of:

(a) 7555 5-vectors of positive random integers,
(b) 7555 5-vectors of positive random coprime integers,
(c) 7555 transverse 5-vectors of positive random coprime integers,
(d) 7555 Calabi-Yau 5-vectors.

These datasets were specifically constructed so as not to form a filtration, therefore at each stage the dataset generated was ensured to not include data which satisfies the additional conditions at the next level. To clarify, each 5-vector in set (a) had weights which shared a common factor, in set (b) all 5-vectors did not satisfy condition 2, and those in set (c) where not in the CY list of (d).

To introduce a consistency across the datasets, all the 5-vectors entries are sorted in increasing order. Initially the weights for each of the datasets (a–c) were sampled using a discretized uniform distribution, $U(1, 2000)$, bound above by 2000 to mimic the highest value in the CY dataset of 1743. However as shown in Fig. 1(a) the weights follow a distribution far from that of a uniform distribution. Therefore to make the generated data more representative, an exponential distribution was fitted to the histogram of all weights in the CY dataset, as shown in Fig. 2. Fitting was performed using the SCIPY library.

This exponential distribution was instead then used to sample weights, and as can be seen in Fig. 3, the frequency distributions of the weights for each of the datasets align much closer to that of the CY data. For reference the weight histograms are shown for the uniform distribution sampling in Appendix A 1.

Aside: Coprimality It is interesting to note that the probability of k randomly chosen integers being coprime is: $1/\zeta(k)$; via the Riemann zeta function. Hence the probability of a random 5-vector of integers being coprime is ~0.964, and therefore the dataset (b) is relatively more common than the dataset (a). Effectively it is easy to randomly produce weighted projective spaces.

B. Principal component analysis

Datasets of vectors can be analysed for hidden structures through the use of principal component analysis (PCA). This method, generally considered to be the simplest form of unsupervised machine-learning, diagonalizes the data's

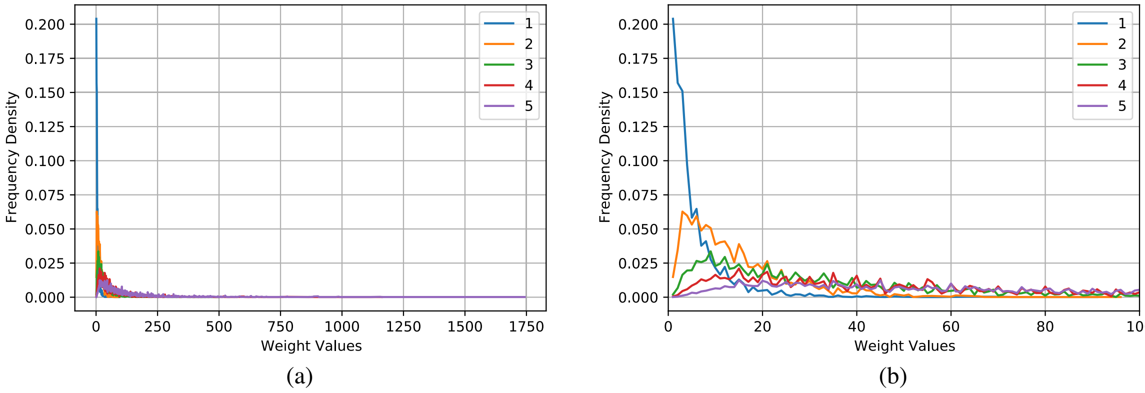


FIG. 1. Frequency distribution of each of the CY 5-vector weights, w_i (labeled by i: 1–5). Figure (b) shows the same data as (a), but restricted to lower entries so as to highlight the low value behavior, due to the entry sorting.

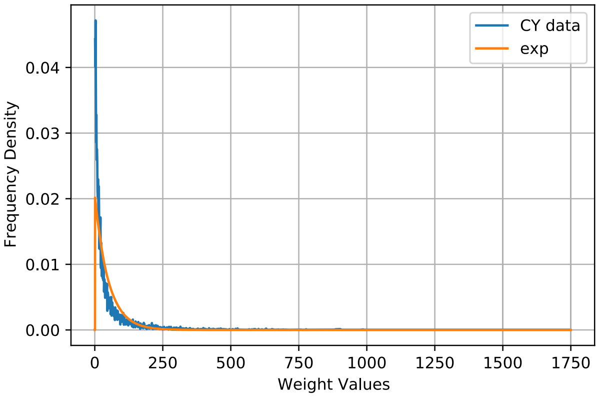


FIG. 2. Frequency distribution for *all* weights occurring across all 5-vectors in the CY dataset. Plot also shows the fitted exponential distribution, with scale parameter 49.536 (to 3 decimal places).

covariance matrix and sorts the respective eigenvalues and eigenvectors.

The covariance matrix of a dataset, computes the covariance between each pairing of the constituent vector elements, defined as:

$$K_{ij} = E(w_i - E(w_i)) \cdot E(w_j - E(w_j)). \qquad (3.1)$$

Since our weight entries are within the field of integers the covariance matrix is symmetric. Diagonalizing this matrix finds the orthogonal linear combinations of the vector entries which dictate the directions of largest variance. Therefore the result of this diagonalization is to identify the data's principal components, which are then sorted in decreasing order according to their respective eigenvalues. The first component then gives the direction where the data has the most alignment and hence the highest variance, with successive decreasing variance until the final entry gives the direction along which the data has the lowest variance.

The structure of the dataset can then be most easily observed through consideration of these principal components. In this study PCA was applied to each of the datasets under consideration independently.

In each case the variance eigenvalues were at least 5 times larger for the first principal component compared to the others. In particular for the CY dataset the first principal component was 2 orders of magnitude larger than the others. This indicates that much of the variation, and hence data structure, is dominated by a single dimension.

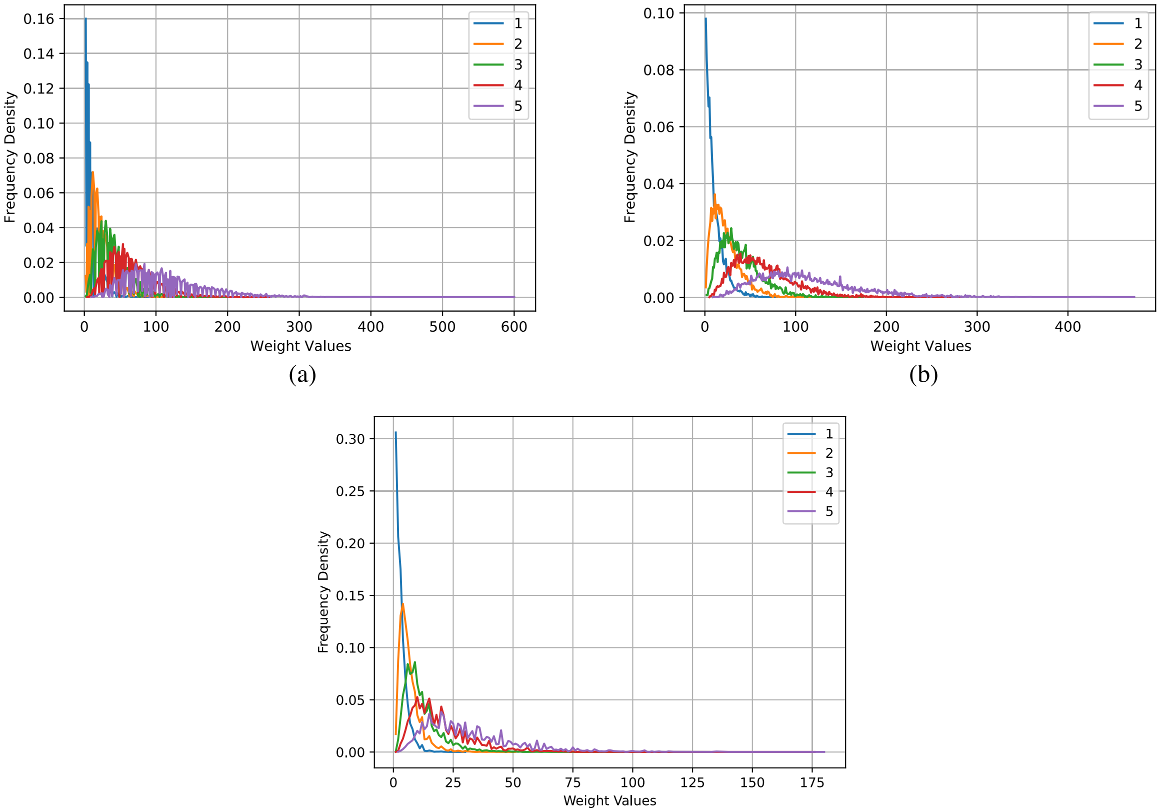


FIG. 3. Frequency distributions for 5-vector weights, w_i (labeled by $i: 1-5$), for each of the generated datasets. The weights were generated using an exponential distribution fitted to the CY weight data; and hence distributions show similar behavior across all datasets, and to the CY dataset. (a) Random integers (b) Random coprime integers (c) Random transverse coprime integers.

Usually a scaling is applied to the data prior to the PCA. The "scaling" process both centres and scales the data such that each entry (i.e., weight) has mean value 0 and standard deviation 1 across the dataset; hence replacing each weight by its respective standardised score. However for this analysis scaling was not used since the data's general structure is based on the relative sizes between the weights (which are sorted according to their size). These relative sizes between weights across each vector are lost through the scaling process, which scales each weight independent of the rest of the vector.

As the data is not scaled one may think that the latter weights of each vector would dominate the behavior (since the weights are ordered). This would lead the covariance matrix to be near-diagonal, and the principal components would align closely to the original vector entries. However, as shown by the covariance matrix for the CY dataset in Eq. (3.2), the matrix is not diagonal and the eigenvectors have significant contribution from multiple components.

$$K_{\mathrm{CY}} = \begin{pmatrix} 41 & 43 & 109 & 250 & 404 \\ 43 & 119 & 278 & 642 & 1017 \\ 109 & 278 & 1795 & 3626 & 5562 \\ 250 & 642 & 3626 & 8588 & 12941 \\ 404 & 1017 & 5562 & 12941 & 20018 \end{pmatrix},$$

$$\varepsilon_{\mathrm{CY}} = \begin{pmatrix} 0.016 & 0.041 & 0.229 & 0.531 & 0.815 \\ 0.021 & 0.036 & -0.973 & 0.100 & 0.205 \\ 0.120 & 0.206 & 0.034 & -0.823 & 0.514 \\ 0.417 & 0.875 & 0.023 & 0.173 & -0.172 \\ 0.900 & -0.435 & 0.003 & 0.018 & -0.008 \end{pmatrix},$$

$$\lambda_{\mathrm{CY}} = \begin{pmatrix} 30071 \\ 233 \\ 161 \\ 74 \\ 21 \end{pmatrix}, \tag{3.2}$$

for eigenvectors as rows of $\varepsilon_{\mathrm{CY}}$ with respective eigenvalues in λ_{CY}; where covariance and eigenvalue entries are given to the nearest integer, and eigenvector entries to 3 decimal places. This implies that the PCA structure is more subtle than a trivial projection. The covariance matrices, eigenvectors and eigenvalues for the other datasets are provided for comparison in Appendix A 2.

To relatively compare the datasets' PCAs, the normalized vectors of eigenvalues are given in Eq. (3.3), for the random "R," coprime "C," transverse "T," and Calabi-Yau "CY" datasets respectively. They show that the first component significantly dominates, and hence lower dimensional representation of the data through PCA will usefully depict the data's underlying linear structure.

$$\lambda_{\mathrm{R}} = \begin{pmatrix} 0.75534 \\ 0.16297 \\ 0.05274 \\ 0.02059 \\ 0.00837 \end{pmatrix}, \qquad \lambda_{\mathrm{C}} = \begin{pmatrix} 0.74845 \\ 0.16856 \\ 0.05417 \\ 0.01997 \\ 0.00885 \end{pmatrix},$$

$$\lambda_{\mathrm{T}} = \begin{pmatrix} 0.91388 \\ 0.04211 \\ 0.02578 \\ 0.01334 \\ 0.00489 \end{pmatrix}, \qquad \lambda_{\mathrm{CY}} = \begin{pmatrix} 0.98399 \\ 0.00764 \\ 0.00525 \\ 0.00242 \\ 0.00070 \end{pmatrix}. \tag{3.3}$$

Hence for the sake of visualization, the first 2 components of each datapoint's principal component projection are plotted as a 2-dimensional scatter diagram for each dataset. These components show the directions with the most variation, and hence display the underlying structure most clearly. The 2d PCA plots are given in Fig. 4, for each of the 4 datasets considered.

The conelike bounding structure of all plots shows the effects of the weight ordering. This is simply that as the first component's value increases (most correlated to the largest, and hence last, weight in the 5-vector) the range of values the second component (roughly correlated to the second-largest/second-last weight) can take increases. Or put more simply, the second-last weight takes values up to the size of the last weight and so this places conelike bounds on the plots. All plots also show higher densities at lower values of the principal components which is also related to this effect.

The PCA plots show that as more of the necessary conditions are added to the datasets, more structure is apparent in the projected outputs. First note, the coprime condition causes a negligible change to the distribution of weights. The transverse condition however has a significant effect. The second components become much more limited and the data begins to separate into approximately two forks. Most exciting, is the jump to the full Calabi-Yau data. Now the PCA shows a clear clustering of the 5-vectors at higher values of the first principal component. This distinct separation into clear lines of datapoints shows a rich structure to the weights of Calabi-Yau projective spaces, which is not present for spaces with just the transverse condition. The reasons for this separation are unclear, however we make conjectural statements about a potential relation to the spaces' Hodge numbers due to a similar structural separation in Sec. III D.

A final note is that the PCA used here was explicitly linear, and hence probes the simplest kind of implicit structure. More technically involved methods of principal component analysis involve kernel methods, called "kernel-PCA." Kernel methods were also used to analyse these datasets, for a variety of traditional kernels (including

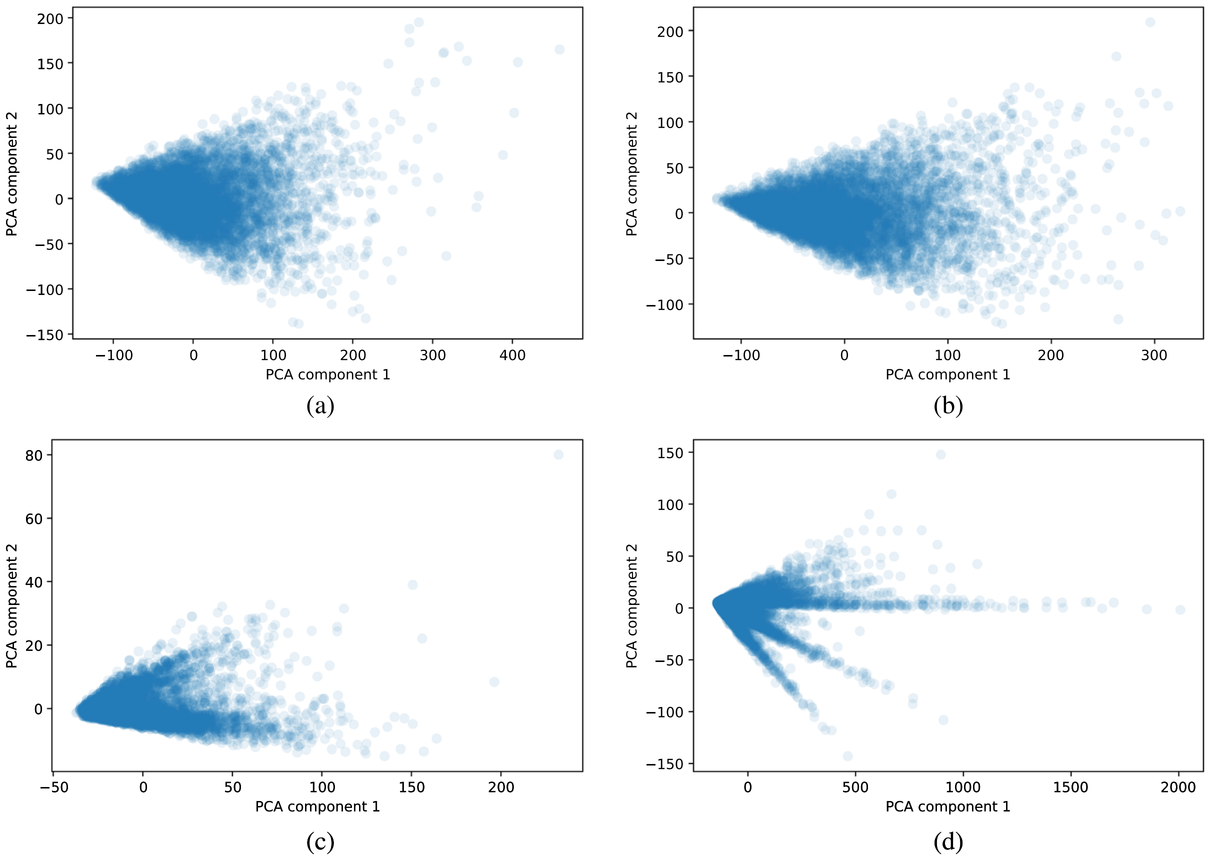


FIG. 4. 2d PCA plots for the 4 considered datasets. As more of the conditions are added, more structure appears, in particular there is some form of distinct class separation for the CY weights. (a) Random integers (b) Random coprime integers (c) Random transverse coprime integers (d) CY weights.

Gaussian, sigmoid, and an array of polynomial kernels), and functionality for this is provided in the respective code scripts. However, none of these methods produced as distinct a clustering separation as that for the linear kernel. Indicating, that surprisingly, the most prominent implicit structure of the Calabi-Yau weights takes a linear form.

C. Topological data analysis

Principal component analysis allows for a 2d visualization of the 5d CY data. Through the PCA with linear kernel, a linear clustering structure was uncovered in the data. To visualise the extension of this behavior to the full 5d space we turn to tools from topological data analysis; specifically *persistent homology*.

The persistent homology of a dataset is constructed through a filtration of Vietoris-Rips complexes. The full CY dataset is first plotted in $\mathbb{R}^5$ with each weight a coordinate, such that each weighted-$\mathbb{P}^4$ is now represented by a point (0-simplex) in the $\mathbb{R}^5$ space due to its respective 5-vector.

5-sphere's of radius d are then drawn around each point, and the range of d values are taken from $0 \longmapsto \infty$. Initially all the spheres will be independent with no overlap, but as d increases the spheres will begin to overlap more frequently. The complex is then constructed by drawing an n-simplex between n points where all their spheres overlap.

Therefore as d increases more simplices are added to the complex, and at each d-value where there is a change to the complex we have a stage in the complex's filtration. The complex hence grows up until a point where all possible simplices lie in the complex. This is where the filtration terminates (no further changes as $d \longmapsto \infty$).

The role of persistent homology in the analysis of this filtration is to examine how long cycles of n-simplices last throughout the filtration before they become filled by the $(n+1)$-simplices they bound. Specifically H_n examines how long cycles of n-simplices exist until becoming filled by $(n+1)$-simplices.

This persistent homology for the CY data was computed for H_0 and H_1 (higher H_n up to $n = 4$ can be computed in 5d space but are incredibly computationally expensive in terms of memory for $n \geq 2$). The persistence diagram for this analysis is shown in Fig. 5, where the diagram plots all members of H_0 and H_1 as points with their respective d values of birth (cycle creation) and death (cycle filling). For specific computation of the persistent homology the python package "ripser" was used [57]; while to review previous

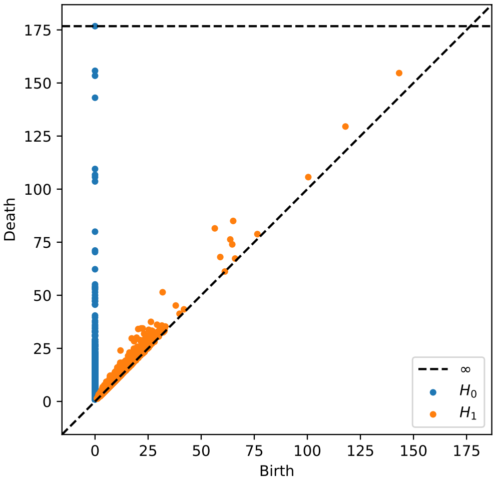


FIG. 5. Persistent diagram for the H_0 and H_1 homology groups of the CY data's Vietoris-Rips complex filtration.

application of these techniques to the string landscape please see [58,59].

As can be seen from the diagram all the members of H_0 are blue points born at $d = 0$, these are each of the 0-cycles (i.e., 0-simplices / datapoints) that exist until they are connected by an edge (1-simplex) to any other datapoint. The behavior shows that there are some datapoints that are significantly far away from the rest of the data and hence join/die much later in the filtration. These points are those with large weight values such that they are far from the origin in the $\mathbb{R}^5$ embedding.

Conversely all members of H_1 are points in orange, and as expected all these 1-cycles (i.e., cycles of 1-simplices/edges which are not boundaries of 2-simplices/triangles) lie close to the diagonal line in the persistence diagram. This behavior indicates a short life of each cycle, a behavior typical of noise in the dataset. Since traditionally it is only points far from the diagonal that indicate significant persistent structure, there is hence not higher dimensional structure formation or nontrivial topology in the data which would deter from the linear clustering behavior seen through the PCA.

D. Analysis of CY topological properties

In addition to the weights used to represent these Calabi-Yau weighted projective spaces, the nontrivial Hodge numbers, $\{h^{1,1}, h^{2,1}\}$, for the surfaces are also provided with the KS databases [46], and repeated with this study's GitHub.

This provides more information for analysis the spectrum of CY weights. Simple plotting of these weights produces an astonishingly familiar structure, one which is exemplified best when the CY's Hodge numbers are plotted against the final (and hence largest) weight, as shown in Fig. 6.

The behavior in Fig. 6(a) shows a similar form of forklike splitting of the datapoints as in the PCA of Fig. 4(d), even with a central fork particularly more dominant than the others. This seemingly linear behavior between final weight and $h^{1,1}$ is quite surprising, and here again the CY hypersurfaces appear to be separating themselves into classes, according to the ratio of $h^{1,1}$ to the final weight, w_5. On the contrary, the behavior in Fig. 6(b), follows the familiar mirror symmetry plot [45], complimenting the linear behavior with $h^{1,1}$ such that their combination will preserve this structure.

Similar behavior also occurs for the other weights in the 5-vectors, despite less obvious clustering. Plots of these relations are given in Appendix A 3.

To further examine this clustering phenomena we plot a histogram of the ratio $h^{1,1}/w_5$ in Fig. 7. Note for this plot only data points with $w_5 > 250$ were used since this was where the class separation was more prominent such that the cluster identification would be improved. As can be seen from the peaks in the figure, there is a clear clustering behavior. Therefore we reexamine this data of ratios with the use of K-means clustering.

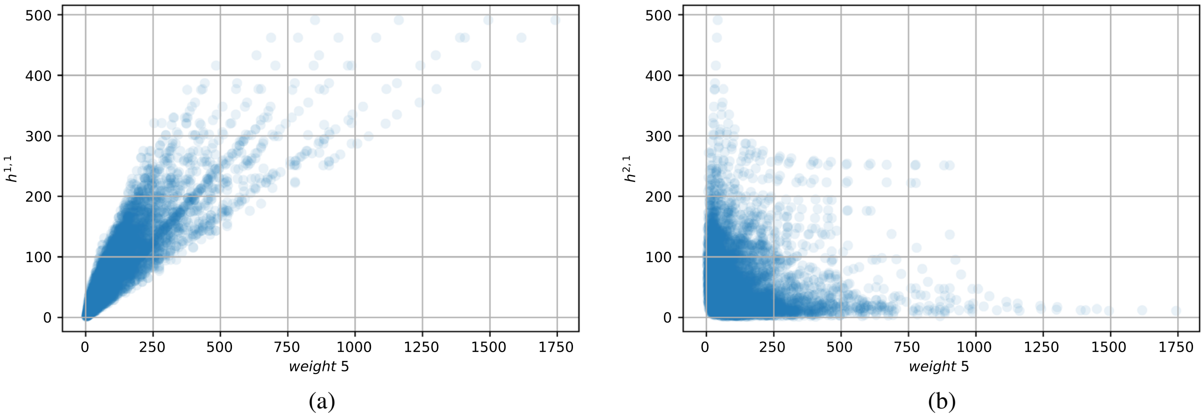


FIG. 6. Distribution of Calabi-Yau weighted projective spaces, according to their final (and largest) weight and (a) $h^{1,1}$ or (b) $h^{2,1}$ respectively.

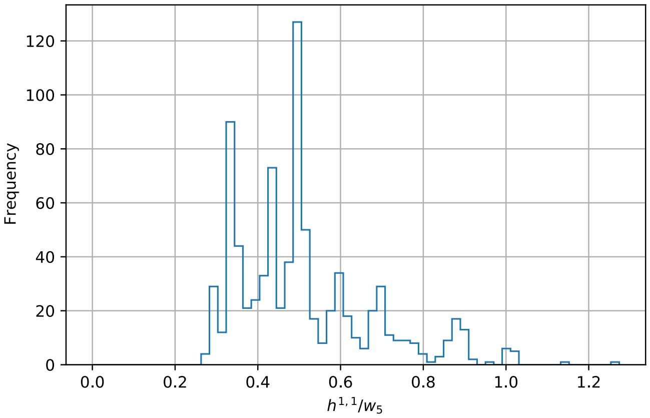


FIG. 7. Frequency of the ratio between $h^{1,1}$ and the largest weight, w_5, for the CY data with $w_5 > 250$ (where structure more prominent). Peaks indicate a natural clustering.

1. Clustering for $h^{1,1}$ classes

As motivated by the formation of a set of linear relationships between w_5 and $h^{1,1}$ shown in Fig. 6(a), and the peak occurrence in the histogram of ratios in Fig. 7, unsupervised clustering methods were used to examine this behavior.

The "outer" ratio data used to produce the histogram plot, where clustering was more prominent, provides a very suitable database for 1-dimensional clustering. The method used was K-means clustering, which takes an input predefined number of clusters, initializes mean values for each cluster, and iteratively updates these means such that the final sum of squared distances from each datapoint to its nearest cluster's mean is minimized. This measure is known as the K-means inertia,

$$\mathscr{I} = \sum_{\mathscr{C}} \sum_{i \in \mathscr{C}} (\mu_{\mathscr{C}} - r_i)^2 \tag{3.4}$$

for clusters, $\mathscr{C}$, with respective means, $\mu_{\mathscr{C}}$, and all data points, i, exclusively in their nearest cluster with ratios, r_i.

Determining the optimal number of clusters to use is a standard problem in K-Means clustering, to motivate this choice we use a novel measure we call "scaled-max-inertia." This measure identifies the maximum squared-distance any point is from its closest cluster center, normalizes it according to that maximum squared-distance from using only one cluster, and adds a weight factor to penalize using an excessive number of clusters. We define this to be

$$\mathscr{I}_{\max} = \frac{\mathrm{Max}_i(\mu_{\mathscr{C}} - r_i)^2}{\mathrm{Max}_i(\mu_1 - r_i)^2} + \frac{(k-1)}{100}, \tag{3.5}$$

where Max_i determines the maximum over all ratios, r_i, examining the squared distance to either the closest cluster's mean, $\mu_{\mathscr{C}}$ or the single cluster's mean, μ_1; then weighting by the number of clusters, k. A plot of scaled-max-inertia against number of clusters identifies an optimum of 10 clusters, as shown in Fig. 8.

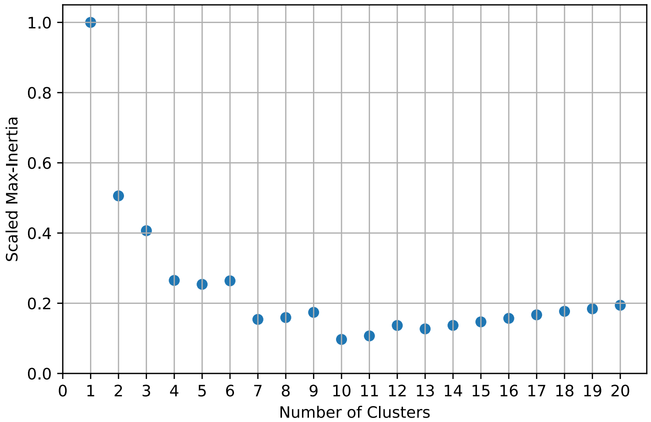


FIG. 8. Plot of scaled max-inertia as the number of clusters used for K-means clustering varies. The minimum identifies an optimum number of clusters: 10.

Using the optimal number of 10 clusters, the separation matches up exceptionally for the outer data, as shown by plots of the cluster bounds in Fig. 9. The clusters sizes for the clusters moving anticlockwise about the plot, for increasing ratio, are $[103, 354, 454, 734, 626, 623, 643, 895, 1419, 1704]$, highlighting that there is a greater density of points at low w_5 as expected, since this was why outer data was focused on for clustering.

To measure the clustering performance we use the standard Inertia measure over the *full* dataset, however normalized by the number of ratios across the dataset, $\hat{\mathscr{I}}$, and an equivalent measure also normalized by the range of the ratios:

$$\hat{\mathscr{I}} = 0.0266, \frac{\hat{\mathscr{I}}}{\max(r_i) - \min(r_i)} = 0.00084, \tag{3.6}$$

These values show that clustering performed exceptionally well, as each ratio in the full CY dataset was less than 0.1% of the ratio-range away from its nearest cluster. Therefore confirming the distinct *linear* behavior observed, as well as the class separation. The distinct classes of CY 5-vectors are also provided in the GitHub.

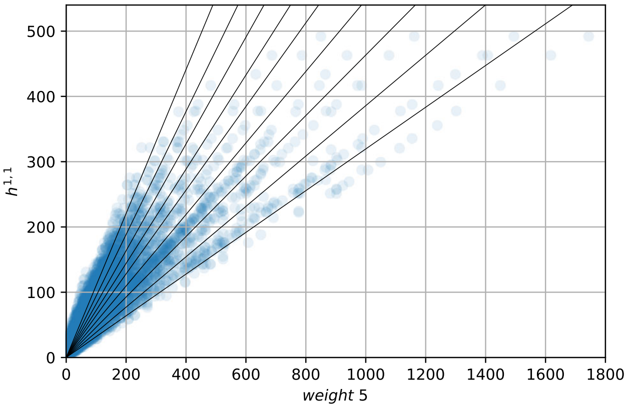


FIG. 9. Plot of the bounds of the 10 clusters produced on the outer data ($w_5 > 250$) via K-means clustering.

IV. MACHINE LEARNING

After use of unsupervised ML methods in Sec. III, we now turn to use of supervised ML methods for learning of the topological properties, as well as the CY property.

A. Architectures

The problems addressed by supervised ML in this study fit into both of the field's typical styles: regression, and classification.

The first set of problems learnt in Sec. IV B learn the topological Hodge numbers (and related Euler number) from the CY 5-vectors of weights. Since the output Hodge numbers can take a large range of integer values the problem was formulated as a regression problem. For this a multilayer perceptron regressor was used to learn each output from the input weights. This regressor is a type of neural network, and the one used specifically had layer sizes of [32, 64, 32], with ReLU activation, and used the *Adam* [60] optimizer method to minimize a mean-squared error loss. The fitting used a batch size of 200, and ran up to 200 epochs until the tolerance of 0.0001 was reached for loss updating.

The second set of problems considered in Sec. IV C sort to determine which dataset a 5-vector belonged to, either by binary classification between each dataset and the CY dataset, or a multiclassification among all 4 datasets. Since these were classification problems an array of different classifiers were used to perform the learning.

The first classifier was a logistic regressor, as perhaps the simplest form of classifier. This logistic regressor had a tolerance of 1 for learning the weight behavior, a C-value of 100 such that there was a low amount of regularization, and used Newton's method for solving, such that multiclassification could also be performed. The second classifier was a support vector machine with a simple linear kernel, and here a higher regularization due to a C-value 1. The third and final classifier used was a neural network classifier (multilayer perceptron also), this time with the same hyperparameters as the regressor except now with a cross-entropy loss function.

1. Measures

To assess learning performance consistent measures are required. For this, dependent on the problem being a regression or classification, different measures were selected as follows.

Regressors: The most standard regressor measure is mean-squared error, *MSE*, which was used for the regressor loss function. However MSE should be considered in relation to the square of the range of output values to be useful, hence a preferable measure also used was mean-absolute-percentage error, *MAPE*. Although it should be noted MAPE has its own drawbacks where it is incalculable when $y_{\text{true}} = 0$ for any of the date inputs. Both these measures are unbounded above and take optimal value of 0 which indicates perfect prediction.

The final regressor measure used was R^2, this evaluates how well a regressor is performing by comparing the proximity of the predicted output to the proximity of the mean (which would be the prediction for a null model regressor). For this measure 1 is optimal, 0 means that prediction is not better than just predicting the true mean each time, and < 0 means worse than just predicting the mean. The equations and output bounds for these measures are given in Eq. (4.1).

$$\begin{aligned} MSE &= \frac{1}{n}\sum(y_{\text{pred}} - y_{\text{true}})^2 \in [0, \infty), \\ MAPE &= \frac{1}{n}\sum\left|\frac{y_{\text{pred}} - y_{\text{true}}}{y_{\text{true}}}\right| \in [0, \infty), \\ R^2 &= 1 - \frac{\sum(y_{\text{true}} - y_{\text{pred}})^2}{\sum(y_{\text{true}} - y_{\text{truemean}})^2} \in (-\infty, 1], \end{aligned} \tag{4.1}$$

summing over all predicted, y_{pred}, and true, y_{true}, outputs in the test data. In addition for R^2 the mean of the true values over the test data outputs, y_{truemean}, was also used.

Classifiers: Trained classifiers predict on input test data by assigning them to classes, this leads to a natural sorting of true (row) vs predicted (column) class frequencies over all the test data, arranged into a confusion matrix, *CM*. From the confusion matrix the normalized sum over the diagonal gives the *Accuracy* which is the proportion of test data correctly classified. However simple accuracy has problems associated to bias data, therefore a better measure of learning is Matthew's correlation coefficient, *MCC*. Both these measures have optimum learning with values of 1, where all test data inputs are allocated to their true class. Equations for these two measures used are given in Eq. (4.2).

$$\begin{aligned} \text{CM} &= \begin{pmatrix} TP & FN \\ FP & TN \end{pmatrix}, \\ \text{Accuracy} &= \frac{TP + TN}{TP + TN + FP + FN} \in [0, 1], \\ \text{MCC} &= \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}} \in [-1, 1], \end{aligned} \tag{4.2}$$

for the binary classification case, where generalizations exist for the multiclassification case.

For all problems 5-fold cross-validation was used, whereby 5 independent versions of each architecture were trained and tested on 5 different train:test partitions of the data, and the learning measures then averaged and standard error computed.

B. ML topological parameters

Topological parameters provide key information about a Calabi-Yau manifold which are essential in the computation of physical phenomena when these manifolds are used for superstring compactifications.

This CY subset from weighted $\mathbb{P}^4$s provides a simple scenario whereby the Hodge numbers (and hence Euler number) can be computed directly from the weights of the toric space that the Calabi-Yau is a hypersurface of. Although it should be noted these formulas are quite nontrivial, as discussed in Sec. II.

Both of these formulas, given in Eq. (2), require greatest common divisor computations throughout their evaluation. Machine-learning methods famously perform badly when approximating these styles of equations and so one would expect the simple neural network architecture used here to not be particularly successful.

The results for the machine-learning of the nontrivial Hodge numbers, and the Euler number are given in Table I. The Hodge number data, provided by [46], is also made available on the GitHub with the Calabi-Yau weight data, and from here the Euler numbers can be calculated using $\chi = 2(h^{1,1} - h^{2,1})$.

The results show a surprisingly successful predictive ability for the Hodge numbers and Euler number, particularly with R^2 values exceeding 0.9. The MAPE values show the Hodge numbers are consistently predicted to be only around 20% off from their true values, while the MSE values provide a less physical measure of learning but are included for reference since they were used as the regressor loss.

Considering the complexity of the equation forms in Eq. (2), it is impressive the neural network can learn any correlating behavior for computation of Hodge numbers or Euler number from the weights alone. In addition, the relatively better performance in learning $h^{1,1}$ may be due to the apparent linear relationship to the weights as exemplified in Sec. III D.

C. ML CY property

The conditions for a 5-vector of weights to represent a weighted projective space which can admit a Calabi-Yau hypersurface are highly nontrivial. As discussed in Sec. III A, the necessary conditions of coprimality and transversity are probed through generation of equivalent datasets, with which the CY dataset can be compared.

Due to the exponential generation techniques making these weights more representative, differentiating which

TABLE I. Learning each of the topological parameters from the Calabi-Yau 5-vectors of weights. Note the final column is Euler number $\chi = 2(h^{1,1} - h^{2,1})$, and since it can evaluate to 0 its MAPE value is not defined. Measurement of learning performance uses 5-fold cross-validation to provide an average and standard error on each measure's value.

	Property			
Measure	$h^{1,1}$	$h^{2,1}$	$[h^{1,1}, h^{2,1}]$	χ
R^2	0.9630 ± 0.0015	0.9450 ± 0.0133	0.9470 ± 0.0041	0.9510 ± 0.0023
MAPE	0.1493 ± 0.0027	0.2519 ± 0.0152	0.2375 ± 0.018	…
MSE	166.9 ± 10.0	147.0 ± 35.6	186.9 ± 13.9	1746.1 ± 82.4

TABLE II. Machine-learning results for three different architectures performing binary classification between the CY data and each specified dataset; and in addition multiclassification across all 4 datasets (labeled "All"). Learning is measured using Accuracy and MCC with 5-fold cross-validation to provide an average and standard error on each measure's value.

		Dataset			
Architecture	Measure	Random	Coprime	Transverse	All
Logistic regressor	Accuracy	0.7152 ± 0.0035	0.7199 ± 0.0037	0.7430 ± 0.0065	0.4825 ± 0.0035
	MCC	0.4352 ± 0.0065	0.4467 ± 0.0073	0.5003 ± 0.0121	0.3141 ± 0.0043
Support vector machine	Accuracy	0.7253 ± 0.0029	0.7116 ± 0.0029	0.7464 ± 0.0014	0.4732 ± 0.0070
	MCC	0.4605 ± 0.0054	0.4374 ± 0.0054	0.5174 ± 0.0029	0.3060 ± 0.0078
Neural network	Accuracy	0.9189 ± 0.0037	0.9178 ± 0.0030	0.7575 ± 0.0024	0.5881 ± 0.0048
	MCC	0.8380 ± 0.0073	0.8377 ± 0.0056	0.5306 ± 0.0059	0.4615 ± 0.0072

dataset a 5-vector belongs to is not possible by eye. Therefore it is natural to wish to consider the effectiveness of machine-learning to this classification problem: learning the Calabi-Yau nature.

Introduced in Sec. IV A, three architectures were used to learn to differentiate the Calabi-Yau weights from each of the other datasets: random integers, coprime random integers, and transverse coprime random integers in binary classification problems. Furthermore they were also used to differentiate all 4 datasets in a multiclassification problem.

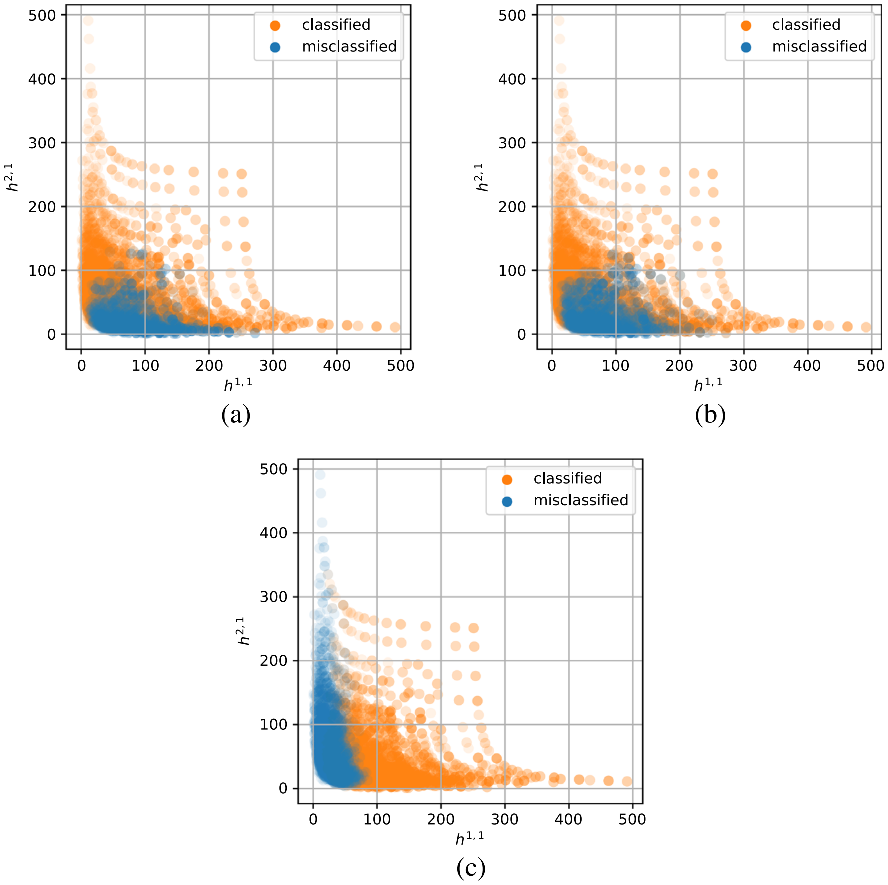
Results for this learning are given in Table II. Measures show that neural networks can well differentiate the Calabi-Yau weights from each of the other datasets. As expected there is minimal difference due to introduction of coprimality, since this is a common behavior for 5-vectors as mentioned in Sec. III A. Once transversity was included into the dataset, the binary classification performance dropped. However performance was still surprisingly good.

A further surprise was the equally good performance of the logistic regressor and support vector machine. These simple architectures could accurately classify approximately three-quarters of the data even without using transversity (where this condition was in both CY and compared dataset).

Multiclassification of all datasets was not as strong. However within these measures the identification of the Calabi-Yau data was considerably better, with most of the performance reduction due to misclassifying between random, coprimality, and transversity. To exemplify this we give a sample confusion matrix for the multiclassification with the logistic regressor:

$$CM_{LR} = \begin{pmatrix} 0.116 & 0.013 & 0.029 & 0.091 \\ 0.076 & 0.083 & 0.074 & 0.020 \\ 0.074 & 0.078 & 0.062 & 0.019 \\ 0.026 & 0.004 & 0.008 & 0.228 \end{pmatrix}, \qquad (4.3)$$

where row indicates true class and column predicted class for each of: random, coprime, transverse, CY respectively. The final entry shows nearly all the Calabi-Yau data is correctly classified (0.25 indicates the full quarter of the accumulated datasets). Therefore measures will indicate lower



(a)

(b)

(c)

FIG. 10. A logistic regressor, trained on 50 CY 5-vectors and 50 non-CY 5-vectors, predicts whether all of the CY 5-vectors are CY or not. The plot shows the distribution of the CY surfaces according to their Hodge numbers. Those in blue are misclassified as non-CY, those in orange are correctly classified to be CY. The non-CY vectors come from datasets of random, coprime, or transverse 5-vectors respectively. (a) Random integers (1899 misclassied) (b) Random coprime integers (1847 misclassied) (c) Random transverse coprime integers (2739 misclassied).

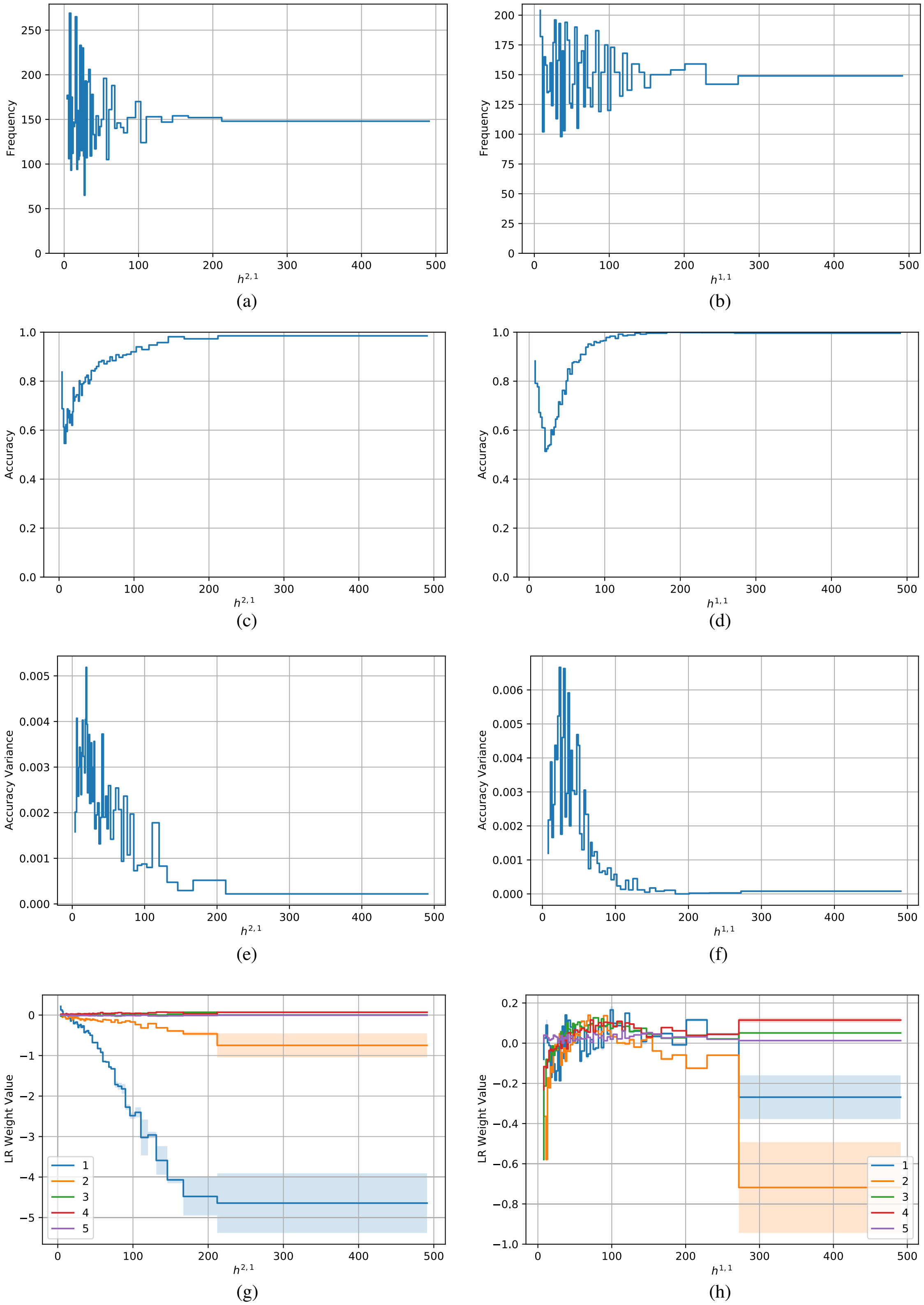


FIG. 11. Relevant plots for logistic regressor learning of the 5-vectors being CY or non-CY. Where the non-CY data was the random data then binning was according to $h^{2,1}$, where it was transverse data then according to $h^{1,1}$. The CY data was binned according to either $h^{1,1}$ or $h^{2,1}$. Figures (a) and (b) show the number of CYs in each Hodge partition bin (half the dataset used in each case as non-CYs cannot be plotted without known Hodge numbers). Figures (c) and (d) show the average accuracies for the LR learning in each case, with (e) and (f) the respective variances (very small comparatively). Finally, figures (g) and (h) show the averaged trained LR weights, plotted with their variances as bands about the average values.

performance where the other conditions cannot be differentiated, and it is likely that these conditions are not the most prominent conditions to indicate the Calabi-Yau property.

To further examine the learning performance we next look explicitly at the misclassifications of the Calabi-Yau data, using again links to the Hodge numbers to identify areas of difficulty.

1. Misclassification analysis with Hodge numbers

Since the logistic regressor performed comparably to the other architectures, and is a significantly simpler architecture than the neural network, its use for misclassification analysis seemed the most appropriate.

Due to the simple structure, only 50 5-vectors in each non-CY dataset were used to train the regressor with another 50 CY 5-vectors. The regressor was then used to predict the class of all the CY data, producing accuracies of: 78%, 81%, 61% when trained with each of the random, coprime and transverse datasets respectively.

Perhaps more curious is the distribution of these CY misclassifications with respect to their Hodge numbers, plotted in Fig. 10. Training random and coprime datasets in both cases leads to perfect classification of CY spaces with high $h^{2,1}$, whereas training with transverse data leads to perfect classification with high $h^{1,1}$.

For reference both other architectures had similar performance with respect to Hodge numbers, as documented in Appendix A 4.

To examine further this relationship, we bin the CY data according to each of the Hodge numbers and only train and test on 5-vectors in each bin's range. This is detailed in Sec. IV C 2.

2. Hodge partitioning

To investigate the dependence of the learning performance on the Hodge numbers, the CY dataset was binned in two independent ways. The first was according to $h^{2,1}$, and the second according to $h^{1,1}$. The bin bounds were optimized such that an approximately consistent number of CYs had Hodge numbers within each bin's bounds, with a preset number of 50 bins used (selected to have a suitable bin size >100). Plots of these bin frequencies are given in Figs. 11(a) and 11(b).

This produced a CY dataset associated to each bin, with which a non-CY 5-vector dataset was randomly sampled. For the $h^{2,1}$ partition the random dataset was used to sample as many non-CY 5-vectors for each bin, such that the datasets were balanced. As training-behavior for the random and coprime datasets was so similar, only the random dataset was used in this investigation. Conversely, for the $h^{1,1}$ partition the transverse dataset was used. These choices of non-CY datasets used for training were selected such that they aligned with the predicted behavior of Sec. IV C 1, where random-training improves high-$h^{2,1}$ performance, and transverse-training improves high-$h^{1,1}$ performance.

For each bin's now balanced dataset an independent logistic regressor (with architecture as before) was initialized, trained, and tested. A random 80% sample of the data was used for training, with testing on the remaining 20% complement. For each bin, the initialization, training, and testing was repeated 20 times, such that variances on the measures could be calculated. Accuracies were recorded for each bin regressor, as well as the final 5 weights used to define the LR.

Accuracies across the bins for both partitions are given in Figs. 11(c) and 11(d), with their respective accuracy variances in Figs. 11(e) and 11(f). There are near perfect predictions at the upper ends of these partitions, with relatively very small variances. Determination of the CY property is hence considerably easier for surfaces whose Hodge numbers take extreme values, and pretraining against data with or without the transverse condition can significantly aid learning depending on what values the Hodge numbers take.

Finally, the 5 averaged LR weights are plotted for each bin (with respective variances surrounding them) in Figs. 11(g) and 11(h). As can be seen by comparing the relative weight sizes, in both cases at the higher ends of the partitions the first two weights particularly dominate the regression. Since each LR weight aligns with the projective space weight, this indicates at these extremes where learning is particular strong, only the first two (i.e., lowest) weights are needed to identify whether the weighted projective space admits a Calabi-Yau hypersurface. Where only the CY dataset has the transversity property (i.e., training against Random) the first weight is the most significant, while where transversity is in both datasets (i.e., training against transverse) the second weight is the most significant.

V. SUMMARY AND OUTLOOK

Through the use of unsupervised machine-learning methods we were able to identify a linear clustering structure of the weighted projective spaces that admit Calabi-Yau hypersurfaces. This structure was first observed through PCA, corroborated with TDA, and then observed again due to relations with the hypersurface's Hodge numbers.

Supervised machine-learning methods then learnt to predict Hodge numbers from the weights directly to a surprisingly exceptional accuracy, perhaps making use of this simple structure. In addition, simple classifier architecture could detect whether a generic weighted-$\mathbb{P}^4$ admitted a Calabi-Yau hypersurface from the weights alone, and with specific pretraining could reach perfect performance at certain extremes of Hodge numbers.

Further analysis into this Calabi-Yau clustering behavior for weighted-$\mathbb{P}^4$s would hope to uncover its source, simultaneously explaining the success of machine-learning techniques on this dataset.

All of the data and code are freely available on GitHub at: [61]. The dataset is readily available at: [62], while another copy is given with this study's scripts on the corresponding GitHub repository.

ACKNOWLEDGMENTS

The authors would like to thank Prof. V. Batyrev for clarifying discussion. D. S. B. is partially supported by Pierre Andurand. Y. H. H. would like to thank STFC for Grant No. ST/J00037X/1. E. H. would like to thank STFC for a PhD studentship.

APPENDIX A

1. Uniformly sampled weight distributions

For reference, the weight frequency distributions for two of the three generated datasets: (a) Random integers, (b) Random coprime integers; as discussed in Sec. III A, are shown below Fig. 12, where the weights were sampled uniformly using a discretization of $U(1, 2000)$.

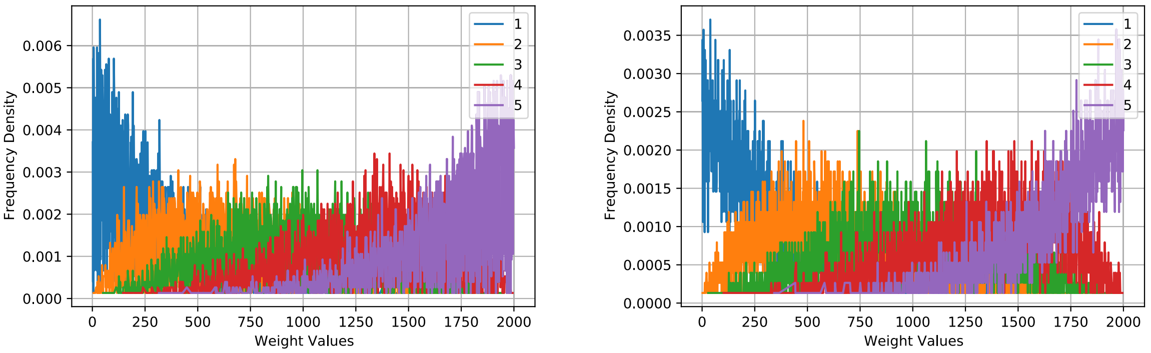
Note the dataset of transverse random coprime integers could not be generated using a uniform distribution. Since the probability of five random integers in this range each dividing another weight negated from the sum is so improbable, no examples were generated running the code for a multiple days on a supercomputing cluster. However generation with an exponential distribution took the order of minutes. Hence the transverse property most likely is has a significant contribution to the exponential weight distribution behavior of the CY data.

2. Additional PCA information

Further to the PCA information provided for the CY dataset in Sec. III B, the covariance matrices, eigenvectors, and eigenvalues are given for the other three datasets here. They are respectively labeled "R" for the random dataset, "C" for coprime dataset, and "T" for transverse dataset. The covariance matrices, K, and eigenvalues, λ, are given to the nearest integer, while eigenvectors, rows of ε, are given to 3 decimal places.

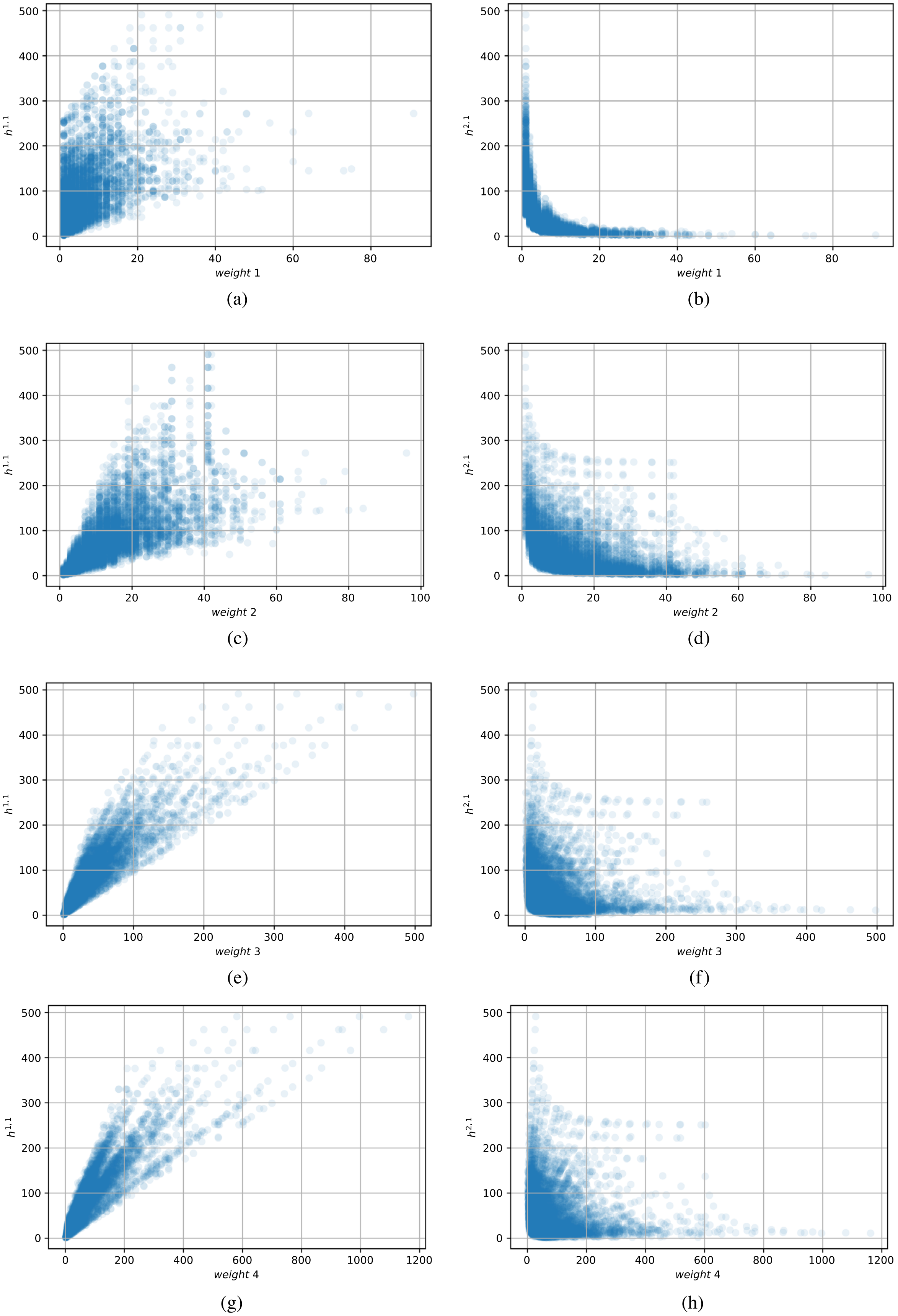
$$K_R = \begin{pmatrix} 97 & 98 & 98 & 96 & 107 \\ 98 & 251 & 250 & 245 & 255 \\ 98 & 250 & 530 & 514 & 542 \\ 96 & 245 & 514 & 1122 & 1157 \\ 107 & 255 & 542 & 1157 & 3614 \end{pmatrix}, \quad \varepsilon_R = \begin{pmatrix} 0.039 & 0.094 & 0.191 & 0.375 & 0.902 \\ -0.121 & -0.298 & -0.519 & -0.669 & 0.424 \\ -0.253 & -0.517 & -0.520 & 0.626 & -0.085 \\ -0.469 & -0.591 & 0.640 & -0.145 & 0.006 \\ -0.837 & 0.535 & -0.117 & 0.006 & 0.003 \end{pmatrix}, \quad \lambda_R = \begin{pmatrix} 4241 \\ 915 \\ 296 \\ 116 \\ 47 \end{pmatrix}, \tag{A1}$$

$$K_C = \begin{pmatrix} 100 & 100 & 101 & 91 & 89 \\ 100 & 254 & 255 & 254 & 249 \\ 101 & 255 & 527 & 534 & 527 \\ 91 & 254 & 534 & 1166 & 1163 \\ 89 & 249 & 527 & 1163 & 3418 \end{pmatrix}, \quad \varepsilon_C = \begin{pmatrix} 0.036 & 0.098 & 0.199 & 0.400 & 0.889 \\ -0.124 & -0.297 & -0.514 & -0.657 & 0.448 \\ -0.284 & -0.532 & -0.497 & 0.617 & -0.095 \\ -0.457 & -0.570 & 0.662 & -0.168 & 0.009 \\ -0.833 & 0.543 & -0.109 & -0.003 & 0.000 \end{pmatrix}, \quad \lambda_C = \begin{pmatrix} 4091 \\ 921 \\ 296 \\ 109 \\ 48 \end{pmatrix}, \tag{A2}$$



(a) (b)

FIG. 12. Frequency distributions for 5-vector weights, w_i (labeled by $i: 1-5$), for the generated datasets of random integers and random coprime integers. Weights were generated using a discretized uniform distribution, $U(1, 2000)$. Distributions show a spread across the range (accounting for the sorting), and hence do not well mimic the CY dataset. (a) Random integers (b) Random coprime integers.



(a)

(b)

(c)

(d)

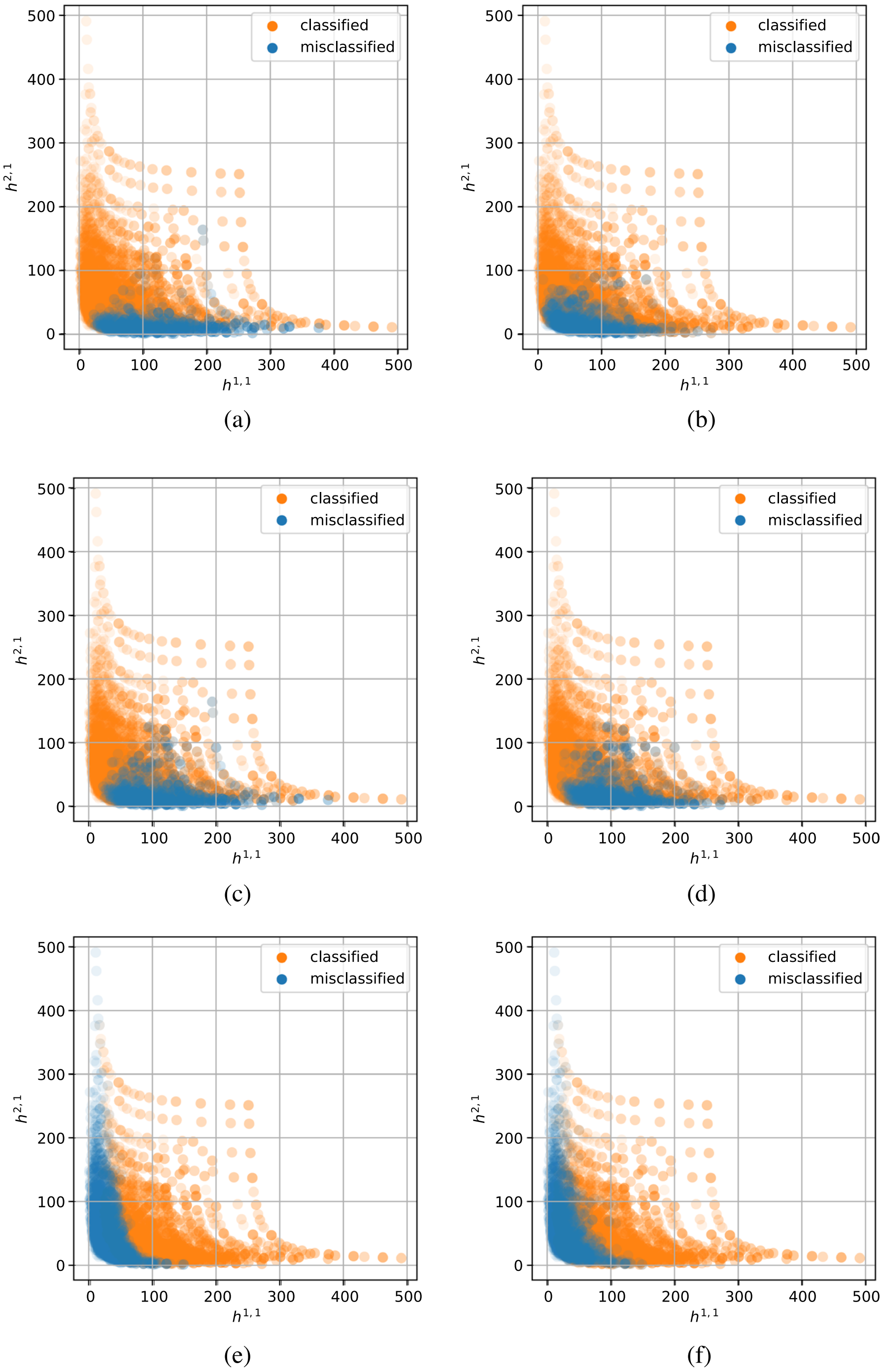
(e)

(f)

(g)

(h)

FIG. 13. Plots of the nontrivial Hodge numbers $\{h^{1,1}, h^{2,1}\}$ against each of the first 4 weights in the CY 5-vectors. Behavior is similar to that with the final weight, showing a linear relationship to $h^{1,1}$ and a relationship preserving the mirror symmetry structure for $h^{2,1}$.



(a)

(b)

(c)

(d)

(e)

(f)

FIG. 14. Classified and misclassified CY 5-vectors plotted with respect to Hodge numbers, where prediction was performed by either of the architectures: support vector machine (SVM), or neural network (NN); trained with each of the non-CY datasets respectively. (a) SVM trained with random (b) NN trained with random (c) SVM trained with coprime (d) NN trained with coprime (e) SVM trained with Transverse (f) NN trained with transverse.

$$K_T = \begin{pmatrix} 6 & 7 & 8 & 12 & 19 \\ 7 & 20 & 25 & 35 & 55 \\ 8 & 25 & 62 & 85 & 125 \\ 12 & 35 & 85 & 173 & 246 \\ 19 & 55 & 125 & 246 & 417 \end{pmatrix}, \quad \varepsilon_T = \begin{pmatrix} 0.040 & 0.114 & 0.264 & 0.507 & 0.812 \\ 0.102 & 0.332 & 0.712 & 0.349 & -0.501 \\ 0.198 & 0.467 & 0.321 & -0.746 & 0.286 \\ -0.428 & -0.660 & 0.556 & -0.253 & 0.091 \\ -0.875 & 0.473 & -0.105 & 0.018 & -0.001 \end{pmatrix}, \quad \lambda_T = \begin{pmatrix} 620 \\ 29 \\ 17 \\ 9 \\ 3 \end{pmatrix}. \tag{A3}$$

3. Additional Hodge plots

Further to the plots of the two nontrivial Hodge numbers of the CY surfaces, $\{h^{1,1}, h^{2,1}\}$, against the final 5-vector weights in Sec. III D, additional plots of these Hodge numbers against the other weights are given here in Fig. 13 for reference.

4. Additional misclassification analysis

Distributions of correctly and incorrectly classified CY 5-vectors for each of the other architectures (support vector machine and neural network), trained on 50 CY and 50 non-CY 5-vectors, are given in Fig. 14. Note the architectures had the same hyperparameters as in previous investigation of Sec. IV C.

The behavior is similar to that for the logistic regressor, where training with random 5-vectors improves determination for high $h^{2,1}$, while training with transverse 5-vectors improves determination for high $h^{1,1}$.

[1] Y. LeCun, Y. Bengio, and G. Hinton, Deep learning, Nature (London) **521**, 436 (2015).

[2] Y.-H. He, Deep-learning the landscape, arXiv:1706.02714.

[3] Y.-H. He, Machine-learning the string landscape, Phys. Lett. B **774**, 564 (2017).

[4] J. Carifio, J. Halverson, D. Krioukov, and B. D. Nelson, Machine learning in the string landscape, J. High Energy Phys. 09 (2017) 157.

[5] D. Krefl and R.-K. Seong, Machine learning of Calabi-Yau volumes, Phys. Rev. D **96**, 066014 (2017).

[6] F. Ruehle, Evolving neural networks with genetic algorithms to study the string landscape, J. High Energy Phys. 08 (2017) 038.

[7] Y.-H. He and M. Kim, Learning algebraic structures: Preliminary investigations, arXiv:1905.02263.

[8] L. Alessandretti, A. Baronchelli, and Y.-H. He, Machine learning meets number theory: The data science of Birch-Swinnerton-Dyer, arXiv:1911.02008.

[9] Y.-H. He, E. Hirst, and T. Peterken, Machine-learning dessins d'enfants: Explorations via modular and Seiberg–Witten curves, J. Phys. A **54**, 075401 (2021).

[10] Y.-H. He and S.-T. Yau, Graph Laplacians, Riemannian manifolds and their machine-learning, arXiv:2006.16619.

[11] Y.-H. He, Machine-learning mathematical structures, arXiv:2101.06317.

[12] A. Davies, P. Veličković, L. Buesing, S. Blackwell, D. Zheng, N. Tomašev, R. Tanburn, P. Battaglia, C. Blundell, A. Juhász *et al.*, Advancing mathematics by guiding human intuition with AI, Nature (London) **600**, 70 (2021).

[13] I. M. Comsa, M. Firsching, and T. Fischbacher, SO(8) supergravity and the magic of machine learning, J. High Energy Phys. 08 (2019) 057.

[14] N. Bobev, T. Fischbacher, and K. Pilch, Properties of the new $\mathcal{N} = 1$ AdS$_4$ vacuum of maximal supergravity, J. High Energy Phys. 01 (2020) 099.

[15] C. Krishnan, V. Mohan, and S. Ray, Machine learning $\mathcal{N} = 8, D = 5$ gauged supergravity, Fortschr. Phys. **68**, 2000027 (2020).

[16] N. Bobev, T. Fischbacher, F. F. Gautason, and K. Pilch, A cornucopia of AdS$_5$ vacua, J. High Energy Phys. 07 (2020) 240.

[17] D. Berman, T. Fischbacher, and G. Inverso, New $N = 1$ AdS$_4$ solutions of type IIB supergravity, arXiv:2111.03002.

[18] D. Berman, T. Fischbacher, G. Inverso, and B. Scellier, Vacua of ω-deformed SO(8) supergravity, arXiv:2201.04173.

[19] P. Candelas, G. T. Horowitz, A. Strominger, and E. Witten, Vacuum configurations for superstrings, Nucl. Phys. **B258**, 46 (1985).

[20] J. Bao, Y.-H. He, E. Hirst, and S. Pietromonaco, Lectures on the Calabi-Yau landscape, arXiv:2001.01212.

[21] Y.-H. He, *The Calabi–Yau Landscape: From Geometry, to Physics, to Machine Learning*, Lecture Notes in Mathematics (Springer, New York, 2021).

[22] P. Candelas, A. M. Dale, C. A. Lutken, and R. Schimmrigk, Complete intersection Calabi-Yau manifolds, Nucl. Phys. **B298**, 493 (1988).

[23] P. Green and T. Hubsch, Calabi-Yau manifolds as complete intersections in products of complex projective spaces, Commun. Math. Phys. **109**, 99 (1987).

[24] K. Bull, Y.-H. He, V. Jejjala, and C. Mishra, Machine learning CICY threefolds, Phys. Lett. B **785**, 65 (2018).

[25] K. Bull, Y.-H. He, V. Jejjala, and C. Mishra, Getting CICY high, Phys. Lett. B **795**, 700 (2019).

[26] S. Krippendorf and M. Syvaeri, Detecting symmetries with neural networks, arXiv:2003.13679.

[27] Y.-H. He and A. Lukas, Machine learning Calabi-Yau four-folds, Phys. Lett. B **815**, 136139 (2021).

[28] M. R. Douglas, S. Lakshminarasimhan, and Y. Qi, Numerical Calabi-Yau metrics from holomorphic networks, arXiv: 2012.04797.

[29] A. Ashmore, R. Deen, Y.-H. He, and B. A. Ovrut, Machine learning line bundle connections, Phys. Lett. B **827**, 136972 (2022).

[30] L. B. Anderson, M. Gerdes, J. Gray, S. Krippendorf, N. Raghuram, and F. Ruehle, Moduli-dependent Calabi-Yau and SU(3)-structure metrics from machine learning, J. High Energy Phys. 05 (2021) 013.

[31] H. Erbin and R. Finotello, Machine learning for complete intersection Calabi-Yau manifolds: A methodological study, Phys. Rev. D **103**, 126014 (2021).

[32] H. Erbin and R. Finotello, Inception neural network for complete intersection Calabi–Yau 3-folds, Mach. Learn. Sci. Tech. **2**, 02LT03 (2021).

[33] H. Erbin, R. Finotello, R. Schneider, and M. Tamaazousti, Deep multi-task mining Calabi–Yau four-folds, Mach. Learn. Sci. Tech. **3**, 015006 (2022).

[34] M. Larfors, A. Lukas, F. Ruehle, and R. Schneider, Learning size and shape of Calabi-Yau spaces, arXiv:2111.01436.

[35] J. Bao, S. Franco, Y.-H. He, E. Hirst, G. Musiker, and Y. Xiao, Quiver mutations, Seiberg duality and machine learning, Phys. Rev. D **102**, 086013 (2020).

[36] J. Bao, Y.-H. He, E. Hirst, J. Hofscheier, A. Kasprzyk, and S. Majumder, Hilbert series, machine learning, and applications to physics, arXiv:2103.13436.

[37] J. Bao, Y.-H. He, and E. Hirst, Neurons on Amoebae, arXiv:2106.03695.

[38] V. Jejjala, D. K. M. Pena, and C. Mishra, Neural network approximations for Calabi-Yau metrics, arXiv:2012.15821.

[39] C. R. Brodie, A. Constantin, A. Lukas, and F. Ruehle, Geodesics in the extended Kähler cone of Calabi-Yau threefolds, arXiv:2108.10323.

[40] A. Cole, S. Krippendorf, A. Schachner, and G. Shiu, Probing the structure of string theory vacua with genetic algorithms and reinforcement learning, in *Proceedings of the 35th Conference on Neural Information Processing Systems* (2021), arXiv:2111.11466.

[41] J. Halverson, Building Quantum Field Theories out of Neurons, arXiv:2112.04527 [Phys. Rev. Lett. (to be published)].

[42] X. Gao and H. Zou, Applying machine learning to the Calabi-Yau orientifolds with string vacua, Phys. Rev. D **105**, 046017 (2022).

[43] A. Cole, A. Schachner, and G. Shiu, Searching the landscape of flux vacua with genetic algorithms, J. High Energy Phys. 11 (2019) 045.

[44] S. Krippendorf, R. Kroepsch, and M. Syvaeri, Revealing systematics in phenomenologically viable flux vacua with reinforcement learning, arXiv:2107.04039.

[45] P. Candelas, M. Lynker, and R. Schimmrigk, Calabi-Yau manifolds in weighted P4, Nucl. Phys. **B341**, 383 (1990).

[46] M. Kreuzer and H. Skarke, Complete classification of reflexive polyhedra in four-dimensions, Adv. Theor. Math. Phys. **4**, 1209 (2000).

[47] H. Skarke, Weight systems for toric Calabi-Yau varieties and reflexivity of newton polyhedra, Mod. Phys. Lett. A **11**, 1637 (1996).

[48] J. Bao, Y.-H. He, E. Hirst, J. Hofscheier, A. Kasprzyk, and S. Majumder, Polytopes and machine learning, arXiv:2109.09602.

[49] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. **12**, 2825 (2011).

[50] M. Kreuzer and H. Skarke, No mirror symmetry in Landau-Ginzburg spectra!, Nucl. Phys. **B388**, 113 (1992).

[51] A. Klemm and R. Schimmrigk, Landau-Ginzburg string vacua, Nucl. Phys. **B411**, 559 (1994).

[52] P. Candelas, X. d. l. Ossa, and S. Katz, Mirror symmetry for Calabi-Yau hypersurfaces in weighted and extensions of Landau-Ginzburg theory, Nucl. Phys. **B450**, 267 (1995).

[53] C. Vafa, String vacua and orbifoldized L-G models, Mod. Phys. Lett. A **04**, 1169 (1989).

[54] A. Klemm, B. Lian, S.-S. Roan, and S.-T. Yau, Calabi-Yau four-folds for m- and f-theory compactifications, Nucl. Phys. **B518**, 515 (1998).

[55] V. V. Batyrev, On the stringy Hodge numbers of mirrors of quasi-smooth Calabi-Yau hypersurfaces, arXiv:2006.15825.

[56] V. V. Batyrev and L. A. Borisov, On Calabi-Yau complete intersections in toric varieties, in *Higher Dimensional Complex Varieties* (de Gruyter, Berlin, 2011), pp. 39–66.

[57] C. Tralie, N. Saul, and R. Bar-On, Ripser.py: A lean persistent homology library for python, J. Open Source Softwaare **3**, 925 (2018).

[58] M. Cirafici, Persistent homology and string vacua, J. High Energy Phys. 03 (2016) 045.

[59] A. Cole and G. Shiu, Topological data analysis for the string landscape, J. High Energy Phys. 03 (2019) 054.

[60] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv:1412.6980.

[61] http://hep.itp.tuwien.ac.at/~kreuzer/CY/.

[62] https://github.com/edhirst/P4CY3ML.git.