Quiver mutations, Seiberg duality, and machine learning

Jiakang Bao,[1,*] Sebastián Franco,[2,3,4,†] Yang-Hui He,[1,5,6,‡] Edward Hirst,[1,§] Gregg Musiker,[7,‖] and Yan Xiao[1,8,¶]

[1]*Department of Mathematics, City, University of London, London EC1V 0HB, United Kingdom*
[2]*Physics Department, The City College of the CUNY, 160 Convent Avenue, New York, New York 10031, USA*
[3]*Physics Program, The Graduate School and University Center, The City University of New York, 365 Fifth Avenue, New York, New York 10016, USA*
[4]*Initiative for the Theoretical Sciences, The Graduate School and University Center, The City University of New York, 365 Fifth Avenue, New York, New York 10016, USA*
[5]*Merton College, University of Oxford, Oxford OX14JD, United Kingdom*
[6]*School of Physics, NanKai University, Tianjin, 300071, People's Republic of China*
[7]*School of Mathematics, University of Minnesota, Minneapolis, Minnesota 55455, USA*
[8]*Department of Physics, Tsinghua University, Beijing 100084, China*

(Received 26 July 2020; accepted 27 August 2020; published 15 October 2020)

We initiate the study of applications of machine learning to Seiberg duality, focusing on the case of quiver gauge theories, a problem also of interest in mathematics in the context of cluster algebras. Within the general theme of Seiberg duality, we define and explore a variety of interesting questions, broadly divided into the binary determination of whether a pair of theories picked from a series of duality classes are dual to each other, as well as the multiclass determination of the duality class to which a given theory belongs. We study how the performance of machine learning depends on several variables, including number of classes and mutation type (finite or infinite). In addition, we evaluate the relative advantages of Naive Bayes classifiers versus convolutional neural networks. Finally, we also investigate how the results are affected by the inclusion of additional data, such as ranks of gauge/flavor groups and certain variables motivated by the existence of underlying Diophantine equations. In all questions considered, high accuracy and confidence can be achieved.

DOI: 10.1103/PhysRevD.102.086013

I. INTRODUCTION

A. Preface

Seiberg duality [1] for supersymmetric quantum field theories is one of the most fundamental concepts in modern physics, generalizing the classical electromagnetic duality of the Maxwell equations. In parallel, cluster algebras [2,3] have become a widely pursued topic in modern mathematics, interlacing structures from geometry, combinatorics, and number theory. These seemingly unrelated subjects were brought together in [4–7] in the context of quiver gauge theories realized as world-volume theories of D-brane probing Calabi-Yau singularities. Interestingly, the common theme—quiver Seiberg duality in physics and mutations of cluster algebras in mathematics—emerged almost simultaneously around 1995, completely unbeknownst to the authors of each. It was not until almost a decade later that a proper dialogue was initiated.

Meanwhile, the authors of [8–12] placed the study of quiver gauge theories and toric Calabi-Yau spaces on a firm footing via brane tilings, or dimer models, which are bipartite tilings of the torus. In the mathematics community, cluster algebras have taken a life of their own [13]. Seiberg duality for quiver gauge theories and cluster mutations for quivers have thus allianced a fruitful matrimony. Continued and often surprising interactions between the physics and mathematics have persisted, ranging from quantum field theory (QFT) amplitudes [14,15], to quantization [16], to dualities [17].

Recently, a program of using the latest technology of machine learning and data science to study mathematical structures was launched [18–20]. Indeed, the authors of [18,21–24] introduced the machine learning paradigm to string theory, and [25,26] to symmetries and dualities.

*jiakang.bao@city.ac.uk
†sfranco@ccny.cuny.edu
‡hey@maths.ox.ac.uk
§edward.hirst@city.ac.uk
‖musiker@math.umn.edu
¶steven1025xiao@gmail.com

Methods in neural networks and classifiers have been applied to study diverse problems in physics and mathematics ranging from classifications of Calabi-Yau threefolds [27–30], to flux compactifications in string theory [31], to AdS/CFT [32], to distinguishing elliptic fibrations [33,34], to finding bundle cohomology on varieties [23,35], to knot hyperbolic volumes [36], to distinguishing standard models properties [37–39], to machine learning the Donaldson algorithm for numerical Calabi-Yau metrics [40], to the algebraic structures of groups and rings [41], to dessin d'enfants [42], and to the Birch-Swinnerton-Dyer conjecture in number theory [43], etc.

Given the highly combinatorial nature of quivers and cluster algebras, it is natural to ask whether the machine learning program could be applied to this context. Specifically, one could wonder where in the hierarchy of difficulty, from the least amenable numerical analysis to the most resilient number theory, would quivers and mutations reside. This is thus the motivation of our current work. The paper is organized as follows. After a rapid parallel introduction to Seiberg duality in quiver gauge theories and cluster mutation, from the physics and mathematics point of view in Secs. II A and II B, we proceed in Secs. III to V to study a host of pertinent problems which we will summarize shortly. We conclude in Sec. VI and present some details of the neural networks and their performances over training in the Appendixes.

B. Summary of results

To provide the readers with an idea of the machine learning performance at a glance, we provide here a brief description of the problem styles addressed in this paper, a list of the quivers used to generate the mutation classes examined in the investigations, and a table summarizing the investigations' key results.

1. Data format

The datasets used in these investigations represent each quiver in consideration by its graph-theoretic adjacency matrix (in some investigations with an additional vector structure augmented on). Each investigation has its own dataset of quivers, generated using the SAGE software [44], such that each full dataset is the union of mutually exclusive sets of quiver matrices, where all quivers in each set belong to the same duality class.

Two styles of classification problems are addressed in this paper, and each processes the input quiver data in a different format. The first is binary classification on pairwise data inputs. Here each data input is a pair of matrices, and each pair can be classified as having its two constituent quivers in the same class or not in the same class. On these problems the Naive Bayes (NB) classification method, as described in Appendix A 2, performed best and was hence used. The second problem style is multiclassification directly on the matrices. Here each data input is a matrix, and the matrix is classified into one of the duality classes the classifier is trained on. On these problems convolutional neural networks (NN), as described in Appendix A 3, performed best and were hence used.

Within each investigation fivefold cross validation was used to produce a statistical dataset of measures for the analysis of the classifier's performance. In fivefold cross validation, five independent classifiers are each trained on 80% of the data and validated on the remaining 20%, such that the union of the validation sets gives the full dataset for the investigation. Measures of the classifiers' performance are calculated for each classifier and averaged. In addition, the investigations were also run for varying training/validation percent splits, with results plotted as "learning curves," shown in Appendix B.

2. Quivers considered

Here we list the quivers used to generate the duality classes making up the datasets of the investigations considered in this paper. They are listed with an adjacency matrix representation and are labeled in the form **Q*i***. Different combinations of these quivers (with further Dynkin type examples) were used in each investigation, as listed in the Table I below with "investigation description."

The first three quivers, **Q1**, **Q2**, **Q3**, as well as **Q12**, **Q13**, **Q15**, are finite mutation type under the duality, while the remaining listed here are infinite mutation type. Additionally other Dynkin and finite mutation types were used in investigations, labeled in the standard SAGE quiver package format [45]. These additional quivers considered were either the Dynkin type of various sizes, labeled by the letter and rank of the Dynkin diagram they are equivalent to (with direction added to the edges), or the affine type which corresponds to affine Dynkin diagrams

TABLE I. The quivers considered in machine learning and their training results.

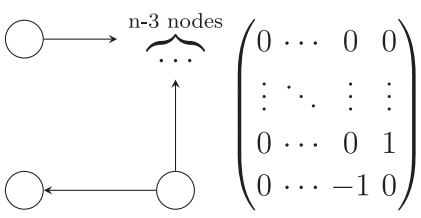
Investigation description	Quivers	Results (acc, ϕ)
NB classification between two mutation classes	['A,' 4]—['D,' 4]	(1.00, 1.00)
	Q4—Q5	(1.00, 1.00)
	['D,' 4]—['A,' (3, 1), 1]	(1.00, 1.00)
	['D,' 4]—Q4	(1.00, 1.00)
NB classification on datasets with varying quiver sizes	Q4—Q5—Q9—Q10	(1.00, 1.00)

(Table continued)

TABLE I. *(Continued)*

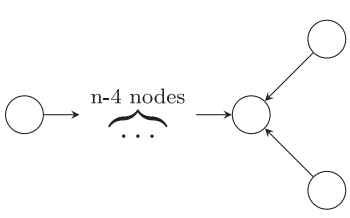
Investigation description	Quivers	Results (acc, ϕ)
NB classification on datasets with more than two mutation classes	['A,' 6]—['D,' 6]—['E,' 6]	(0.90, 0.82)
	['A,' 4]—['D,' 4]—['A,' (3, 1), 1]—['A,' (2, 2), 1]	(0.85, 0.70)
	['A,' 6]—['D,' 6]—['E,' 6]—['A,' 4]—['D,' 4]—['A,' (3, 1), 1]—['A,' (2, 2), 1]	(0.75, ∼)
	Q4—Q5—Q6	(0.91, 0.82)
	Q4—Q5—Q6—Q7	(0.86, 0.72)
	Q4—Q5—Q6—Q7—Q8	(0.84, 0.67)
	['T,' (4, 4, 4)]—['T,' (4, 5, 3)]—['T,' (4, 6, 2)]	(0.89, 0.78)
NB extrapolation predictions: validating on different classes/mutation depths to training	Train ['A,' 6], ['D,' 6]—Valid ['E,' 6]	(0.60, 0.25)
	Train Q4, Q5 low depths -Valid Q4, Q5 high depths	(0.50, 0.00)
	Train Q4, Q5 low & high depths -Valid Q4, Q5 middle depths	(0.65, 0.33)
NB classification on enhanced datasets with rank vectors	Q4—Q5—Q6	(0.91, 0.83)
	Q4—Q5—Q6—Q7	(0.86, 0.72)
	Q4—Q5—Q6—Q7—Q8	(0.84, 0.67)
NB classification on enhanced datasets with Diophantine variables	Q9—Q10—Q11	(0.91, 0.84)
	Q4—Q5—Q6	(0.91, 0.83)
	Q4—Q5—Q6—Q7	(0.86, 0.72)
	Q4—Q5—Q6—Q7—Q8	(0.84, 0.69)
NN classification between finite-type classes	Q12—Q13—Q15	(0.33, ∼)
NN classification on mixed mutation type (finite and infinite)	Q12—Q13- Q14	(0.55, ∼)
NN classification against random antisymmetric matrices	Q9—Antisymm	(0.97, ∼)
NN classification on enhanced datasets with rank vectors	Q12—Q13—Q14	(1.00, ∼)
	Q12—Q13—Q15	(0.71, ∼)
NN extrapolation predictions: validating on different mutation depths to training (with rank vector data enhancement)	Train Q12, Q13, Q14 low depths—Valid Q12, Q13, Q14 high depths	(0.74, ∼)
	Train Q12, Q13, Q14 low & high depths—Valid Q12, Q13, Q14 middle depths	(1.00, 1.00)
	Train Q12, Q13, Q14, Q15 low & high depths—Valid Q12, Q13, Q14, Q15 middle depths	(0.98, ∼)
NN classification against random antisymmetric matrices (with rank vector data enhancement)	Q9—Antisymm	(1.00, ∼)
	Q9—Q10—Antisymm	(0.85, ∼)

and are labeled using Kac's notation with the Dynkin letter, the rank, and an optional twist. In the case of affine A, the rank is given by a pair of integers for the number of clockwise/anticlockwise edges, respectively. The specific affine quiver used to generate a mutation class used in an investigation is the choice autogenerated by the SAGE package for the input label information. Finally, the T type are so named for being shaped like a letter "T," their three integer entries give the number of nodes in each of the branches from the branch point (inclusive). These quivers are described further as they are introduced with each investigation.



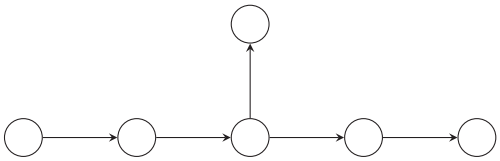
$$\begin{pmatrix} 0 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 1 \\ 0 & \cdots & -1 & 0 \end{pmatrix}$$

Q1: ['A', n]



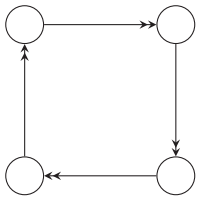
$$\begin{pmatrix} 0 & \cdots & 0 & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & -1 & -1 \\ 0 & \cdots & 1 & 0 & 0 \\ 0 & \cdots & 1 & 0 & 0 \end{pmatrix}$$

Q2: ['D', n]



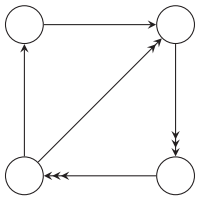
$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 1 \\ 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \end{pmatrix}$$

Q3: ['E', 6]



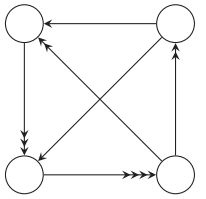
$$\begin{pmatrix} 0 & 2 & 0 & -2 \\ -2 & 0 & 2 & 0 \\ 0 & -2 & 0 & 2 \\ 2 & 0 & -2 & 0 \end{pmatrix}$$

Q4



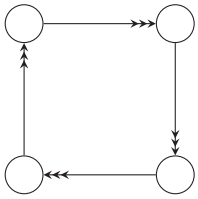
$$\begin{pmatrix} 0 & 1 & 0 & -1 \\ -1 & 0 & 3 & -2 \\ 0 & -3 & 0 & 3 \\ 1 & 2 & -3 & 0 \end{pmatrix}$$

Q5



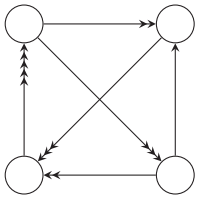
$$\begin{pmatrix} 0 & -1 & -2 & 3 \\ 1 & 0 & -2 & 1 \\ 2 & 2 & 0 & -4 \\ -3 & -1 & 4 & 0 \end{pmatrix}$$

Q6



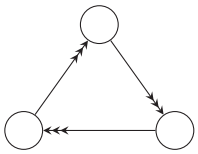
$$\begin{pmatrix} 0 & 3 & 0 & -3 \\ -3 & 0 & 3 & 0 \\ 0 & -3 & 0 & 3 \\ 3 & 0 & -3 & 0 \end{pmatrix}$$

Q7



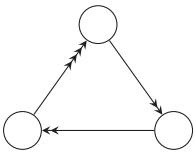
$$\begin{pmatrix} 0 & 2 & 3 & -5 \\ -2 & 0 & -1 & 3 \\ -3 & 1 & 0 & 2 \\ 5 & -3 & -2 & 0 \end{pmatrix}$$

Q8



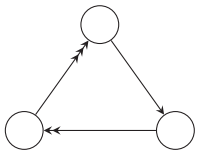
$$\begin{pmatrix} 0 & 3 & -3 \\ -3 & 0 & 3 \\ 3 & -3 & 0 \end{pmatrix}$$

Q9



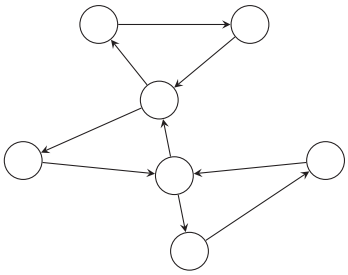
$$\begin{pmatrix} 0 & 2 & -4 \\ -2 & 0 & 2 \\ 4 & -2 & 0 \end{pmatrix}$$

Q10



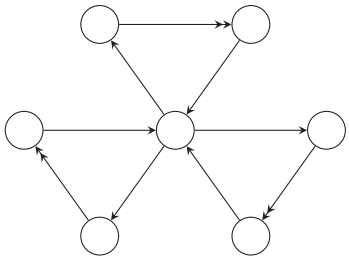
$$\begin{pmatrix} 0 & 1 & -3 \\ -1 & 0 & 2 \\ 3 & -2 & 0 \end{pmatrix}$$

Q11



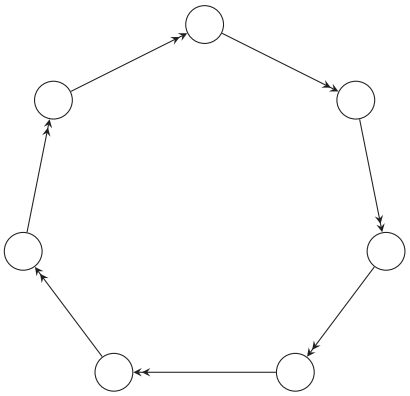
$$\begin{pmatrix} 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 \end{pmatrix}$$

Q12: triangulated 10-gon in the mutation class [‘A’, 7]



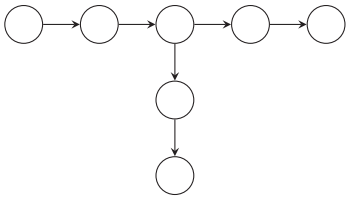
$$\begin{pmatrix} 0 & 2 & -1 & 0 & 0 & 0 & 0 \\ -2 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & -1 & 1 & 1 & -1 \\ 0 & 0 & 1 & 0 & -2 & 0 & 0 \\ 0 & 0 & -1 & 2 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & 0 & -2 & 0 \end{pmatrix}$$

Q13: [‘X’, 7]



$$\begin{pmatrix} 0 & 2 & 0 & 0 & 0 & 0 & -2 \\ -2 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & -2 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & -2 & 0 & 2 \\ 2 & 0 & 0 & 0 & 0 & -2 & 0 \end{pmatrix}$$

Q14



$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 \end{pmatrix}$$

Q15: T-type, ['T', [3,3,3]], which is also of type affine E_6, ['E', 6, 1]

3. Investigation results

Here we tabulate each investigation with a brief description, a list of the quivers used to generate the duality classes in the dataset, and the measures of the learning performance. The measures of performance (as described in Appendix A 4) are presented as a pair: (acc, ϕ), consisting of accuracy of agreement, acc, and Matthews' correlation coefficient, ϕ, where calculated. Both evaluate to 1 for perfect learning, and results are shown to two decimal places.

Dynkin and T type quivers are denoted using the SAGE quiver package convention; other infinite mutation type quivers are denoted using the label assigned in the preceding "quivers considered" list.

NB classifier results showed perfect classification between two mutation type classes. Classifying classes of different quiver sizes was trivial and did not reduce performance as expected. Where classification was between more than two classes, the performance was lower but still very good. Enhancing the datasets with rank information, or Diophantine-inspired variables, did not improve NB classification.

NNs required rank information in their dataset to classify well, but with this included NNs outperformed the NB classifier, particularly when classifying quivers at unseen mutation depths and when classifying against random antisymmetric matrices.

We should also mention that we are using the word "depth" throughout the paper. Starting with a quiver (at depth 0) having n nodes, we have n choices of dualizing one node. These newly generated quivers are said to be at depth 1. We can then apply mutations to these depth-one quivers again by choosing one node to dualize. Such quivers obtained are at depth 2 (except the quiver at depth 0 we start with, i.e., dualizing the same node twice). Hence, when we say a quiver is at depth k, the (shortest) distance would be k from this quiver to our starting quiver under mutations.

II. DRAMATIS PERSONAE

A. Seiberg duality

In this section we review Seiberg duality, which is an IR equivalence between $4d\ \mathcal{N} = 1$ gauge theories [1]. We will phrase our discussion in the language of quivers, since all the theories considered in this paper are of this type.

Let us consider dualizing a node j in the quiver, which does not have adjoint chiral fields.[1] The transformation of the gauge theory can be summarized in terms of the following rules:

1. **Flavors.** In physics, the arrows connected to the mutated node are usually referred to as *flavors*. The flavors transform by simply reversing their orientation, namely:

(1.a) Replace every incoming arrow $i \to j$ with the outgoing arrow $j \to i$. Calling X_{ij} the incoming arrow, we replace it by the dual flavor $\tilde{X}_{ji}$.

(1.b) Replace every outgoing arrow $j \to k$ with the incoming arrow $k \to j$. Calling X_{jk} the outgoing arrow, we replace it by the dual flavor $\tilde{X}_{kj}$.

This is the quiver implementation of the fact that the magnetic flavors are in the complex conjugate representations, of both the dualized gauge group and the spectator nodes, of the original flavors.[2] This transformation is shown in Fig. 1.

2. **Mesons**. Next we add *mesons*, i.e., composite arrows, to the quiver as follows. For every 2-path $i \to j \to k$ we add a new arrow $i \to k$. This meson M_{ik} can be regarded as the composition of the flavors $i \to j$ and $j \to k$ of the original theory, namely $M_{ik} = X_{ij}X_{jk}$. In other words, we generate all possible composite arrows consisting of incom-

[1]Generalizations of Seiberg duality to gauge groups with adjoints are known under certain conditions (see, e.g., [46–48]).

[2]In our discussion, including the points that follow, we allow for the possibility of chiral fields connecting a given pair of nodes in both directions.

FIG. 1. Schematic representation of Seiberg duality. The dualized node j can actually be connected to multiple nodes by incoming and outgoing arrows.

ing and outgoing chiral fields. Figure 1 also illustrates the addition of a meson.

3. **Ranks**. The rank of the dualized node transforms as

$$N'_j = N_{f_j} - N_j, \tag{2.1}$$

where N_{f_j} is the number of flavors at the dualized node j. Later we will consider generic quivers, which are not necessarily anomaly-free. These quivers are interesting from a mathematical point of view and, in such cases, we will not consider the ranks of the nodes. Ranks will only be taken into account for anomaly-free quivers, i.e., theories for which the gauge (and hence dualizable) nodes have an equal number of incoming and outgoing arrows. In these cases,

$$N_{f_j} = N_{\text{in}_j} = N_{\text{out}_j}, \tag{2.2}$$

which, more explicitly, is given by

$$N_{f_j} = N_{\text{in}_j} = \sum_{i\to j} a_{ij} N_i, \tag{2.3}$$

with a_{ij} the (positive) number of bifundamental arrows going from node i into node j.

4. **Superpotential.** The superpotential transforms as follows:

(4.a) In the original superpotential, we replace instances of $X_{ij}X_{jk}$ with the meson M_{ik} obtained by composing the two arrows.

(4.b) Cubic dual flavors–meson couplings. For every meson, we add a new cubic term in the superpotential, coupling it to the corresponding magnetic flavors. Namely, we add the term $M_{ik}\tilde{X}_{kj}\tilde{X}_{ji}$.

If there are fields that acquire mass in this process, we can integrate them out using their equations of motion.

All the rules discussed above, with the exception of the one for the ranks, are the same ones that are used for cluster algebras. Cluster algebras also come equipped with a set of generators known as cluster variables.

B. Mutation of cluster algebras

Mathematically speaking, an algebra is a structure that functions as a vector space with the additional feature that elements can be multiplied together. An algebra can be presented by generators, think of basis vectors, and relations, i.e., algebraic dependencies generalizing linear dependencies of a vector space. A rank n cluster algebra is a subalgebra of the field of rational functions in n variables where its generators can be grouped together into algebraically independent sets known as clusters, also all of size n, such that certain exchange relations allow one to transition from one cluster to another [2]. These exchange relations, known as cluster mutation, can be described using the language of quivers, echoing the description of Seiberg duality in physics.

5. **Cluster variables.** Given an initial cluster $\{x_1, x_2, \ldots, x_n\}$, we allow cluster mutations in n directions, each of the form

$$x_j x'_j = \prod_{i\to j \text{ in Q}} x_i + \prod_{j\to k \text{ in Q}} x_k$$

for each $1 \le j \le n$, and where the products are over all incoming arrows and outgoing arrows, respectively. We thus get a new generator, cluster variable, x'_j, yielding the cluster $\{x_1, x_2, \ldots, x_{j-1}, x'_j, x_{j+1}, \ldots, x_n\}$. The process of cluster mutation may be continued but to mutate while using this new cluster as a reference, we use the quiver $\mu_j Q$ in place of Q, where $\mu_j Q$ is the quiver obtained by applying the rules of Seiberg duality at node j.

Given a quiver Q, we construct the associated cluster algebra $\mathcal{A}_Q$ by applying cluster mutation in all directions and iterating to obtain the full list of cluster variables, i.e., generators of $\mathcal{A}_Q$. Generically, this process yields an infinite number of generators for the cluster algebra, as well as an infinite number of different quivers along the way. However, in special cases, a cluster algebra and its defining quiver have a specified mutation type.

We refer to a cluster algebra, or its associated quiver, as being of finite type if it has a finite number of generators, i.e., cluster variables, constructed by the cluster mutation process.[3] As proven by Fomin and Zelevinsky [3], the list of cluster algebras of finite type exactly agree with Gabriel's ADE classification[4] of quivers admitting only

[3]This is a different statement than saying the cluster algebra is finitely generated, or Noetherian, as an algebra. There are examples of Noetherian algebras that admit an infinite number of cluster variables as generators. The simplest such example corresponds to the quiver associated with $SU(2)$ theories consisting of two nodes and two arrows between them. There are an infinite number of cluster variables for the associated cluster algebra even though as an algebra, it is generated by four elements [49,50].

[4]The ADE's are just the usual label following the label Dynkin diagrams. Or if we allow cluster algebras associated with skew-symmetrizable matrices rather than only quivers, which must be skew symmetric, we get the Cartan-Killing or Dynkin classification including types B, C, F_4, and G_2 as well.

finitely many indecomposable representations [51], or those equivalent to them via quiver mutation, i.e., Seiberg duality.

Another important family of cluster algebras are those of the finite mutation type. Such cluster algebras are those with only a finite number of quivers reachable via mutation, i.e., Seiberg duality. This class of cluster algebras completely encompasses the subclass of cluster algebras of finite type. In totality, this class contains all rank 2 cluster algebras, such as the aforementioned cluster algebra associated with $SU(2)$, cluster algebras of surface type, and 11 exceptional types (E_6, E_7, E_8, affine E_6, E_7, E_8, elliptic E_6, E_7, E_8, and two additional quivers known as X_6 and X_7) [52,53]. Such finite mutation type quivers have also been studied previously in the physics literature where they were referred to as complete quantum field theories [54].

Cluster algebras of surface types, i.e., associated with orientable Riemann surfaces, were first described by Fomin, Shapiro, and Thurston [55]. Generically, the quiver associated with a triangulation of a Riemann surface is obtained by taking the medial graph where nodes of the quiver correspond to nonboundary arcs of the triangulation, and we draw an arrow of the quiver between nodes i and j for every triangular face where arcs associated with i and j meet at a vertex and j follows i in clockwise order. Mutating at a node corresponds to flipping between the two possible diagonals for triangulating a quadrilateral. Since such triangulations live on an orientable Riemann surface, any associated quiver has at most two arrows between any given pair of nodes, thus demonstrating that such cluster algebras admit only finitely many quivers and are hence of the finite mutation type. The 11 exceptional cases of Felikson, Shapiro, and Tumarkin do not have a surface model but at least the finite and affine type E quivers are well-known from previous representation theory, e.g., Gabriel's ADE classification, and Kac's extension to affine quivers [56].

In this paper we will focus on the transformation of the quiver (rules 1 and 2) and in some cases include information on the ranks (rule 3), so we will not deal with rule 4 nor with rule 5. Even with this restriction, we will manage to obtain nontrivial results. Having said that, the superpotential is a crucial element of the duality, as is the mutation of cluster variables in the context of cluster algebras. We plan to incorporate both of these in future studies.

III. RECOGNIZING MUTATIONS

There are various ways to construct the dataset. We can directly assign each mutation class a different label. Then the machine will be asked to do a multiclass classification. We can also have datasets that consist of matrix pairs so that every $\{\text{input} \to \text{output}\}$ has the form

$$\{(M_1, M_2) \to 1/0\}, \tag{3.1}$$

where 1 indicates that M_1 and M_2 are in the same class while 0 indicates that they are not. Let us first start with the latter using the *Mathematica* built-in function CLASSIFY.

A. Classifying two mutation classes

As the simplest example, let us machine learn only two different classes, ['A,' 4] and ['D,' 4],[5] shown with their adjacency matrices as **Q1** and **Q2**, for the cases $n = 4$, in the quivers list of Sec. I B.

Notice that these matrices/quivers are of finite mutation types, i.e., the duality trees are closed. Many (but not all) quivers in finite mutation types[6] contain sources and sinks, and are hence anomalous. Albeit not physically meaningful, we are still interested in these quivers from pure mathematics and machine learning viewpoints. Furthermore, we can compare these results with those from infinite mutation types.

The result[7] of fivefold cross validation is tabulated in Table II. We also plot the learning curves at different training percentages in Fig. 11. We can see that the machine gives 100% accuracy most of the time, which is very inspiring.

Before we add more mutation classes to our data, we are also curious about how the machine would behave when it is asked to predict unseen classes. In the above two-class example, the validation set V is the complement of the training set T. Therefore, what the machine validates are in the same classes as those being trained. Now let us train the ['A,' 6] and ['D,' 6] classes and validate ['E,' 6] (shown as **Q3** in the quivers considered list). In the validation dataset, 1's are always from pairs in ['E,' 6] while 0's are from ['E,' 6]/['A,' 6] or ['E,' 6]/['D,' 6] pairs.[8] The learning result with various training percentages is given in Fig. 2. We find that the overall result is not very satisfying, and the

[5]Henceforth, we will use the same notation as in SAGE [44,45] for known quiver mutation types, and we will not specify the matrices and quivers.

[6]To be clear, we should point out that finite *mutation types* and finite *types* refer to different concepts. In the sense of [3,52], a finite mutation type indicates that there are finitely many dual quivers generated from our starting quiver while a finite type is the namesake of a Dynkin type. Sometimes we will use the term "finite classes." This is the same as "finite mutation types." However, note the word "class" is slightly different from "mutation type" in our context. Each class refers to one duality tree. For instance, ['A,' 4] and ['A,' 5] are not in the same class as they are certainly not duals, but they are both of finite mutation types.

[7]Note the metrics used to evaluate the machine's performance (accuracy, F-score, and MCC ϕ) are defined in Appendix A 4.

[8]Unlike 1's, the 0's always have a matrix from trained classes. However, as we will further study in Sec. III B and Appendix A 2 when finding the optimal method of the classifier, assigning 1 or 0 to a given pair is solely determined by the two matrices in this pair. Any other matrices, no matter whether they are related by mutations to the matrices in this pair, are irrelevant. In this sense, the ['E,' 6]/['A,' 6] and the ['E,' 6]/['D,' 6] pairs are always unseen classes.

TABLE II. Training and validating two classes: ['A,' 4] and ['D,' 4]. We generate (144 + 50) matrices. There are 9026 1's and 7193 0's. The method is chosen by the machine. The results are accurate to the floating point precision, but decimal points are not shown.

Accuracy	F-Score	ϕ
1 ± 0	1 ± 0	1 ± 0

Matthews ϕ could be indeterminate occasionally. From the confusion matrices, we can know that there is still always a zero entry. This zero always appears at false positive (FP) or true negative (TN); i.e., only 1's or only 0's are predicted when the actual values are 0 in each single training. It is reasonable to see that such result as the machine has met some unseen mutation classes. This also shows that the machine is certainly not learning mutations (at least not the whole knowledge thereof) when the dataset contains only two different mutation classes.[9]

B. Fixing the Method

The CLASSIFY function in *Mathematica* has an option where one can specify the method used in the classifier. So far, this value is default in our experiments, and the method is chosen automatically by the machine. However, it is worth finding what method can give better predictions. It turns out that the NB is the method we should choose. When studying the ADE Dynkin type quivers with six nodes above, we find that at each training percentage, the relatively higher accuracy is obtained only when the machine chooses NB. Hence, we perform this experiment with the same dataset again, but this time, we fix our method to NB. The learning curves are reported in Fig. 3. We find that the standard deviations are indeed reduced. The trends of the curves behave like those of the usual learning curves. Moreover, although there is still always a zero entry in the confusion matrix, the Matthews ϕ is *never* indeterminate anymore. In contrast, we can try what happens if we fixate on other methods. As an example, the result of using only random forest with the same dataset at 80% training percentage is reported in Table III. It is obviously inferior to the result using NB. Henceforth, unless specified, we will always apply NB in the CLASSIFY function for future experiments.[10]

[9]One may wonder whether the dimensions of matrices would affect our result, but in fact it is not a main influence. We will further study this when we include more different mutation classes in our training.

[10]We also tried different methods when machine learning the example in Sec. III D which has four different classes. It turns out that in the built-in CLASSIFY function, NB gives nearly 85% accuracy at 80% training percentage while NN gives ~60% accuracy and support vector machine (SVM) gives ~50% accuracy. Moreover, NN and SVM would take 1–2 minutes while NB would only take 1–2 seconds.

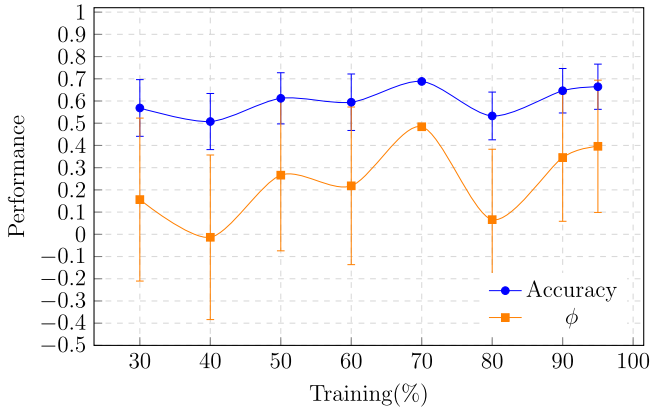


FIG. 2. Training two classes: ['A,' 6] and ['D,' 6], and validating ['E,' 6] (0's from ['E,' 6]-['A,' 6] and ['E,' 6]-['D,' 6] pairs). We generate 517, 572, and 600 matrices, respectively. We choose training data out of 14182 pairs and validation data out of 13897 pairs. Data with indeterminate ϕ's, which appeared several times, are not plotted. These indeterminate ϕ's appeared 7 times in all (training and validation 10 times at each training percentage). The method is chosen by the machine.

Now, we would like to understand why NB always yields such good results. In Appendix A 2, we give a mathematical background of NB. The main reason is that the mutual independence of matrix pairs coincides with the basic assumption of NB.

C. Two classes revisit

To some extent, machine learning finite mutation classes would not be that necessary in application simply because we can traverse all the matrices. Let us try another example which contains two infinite mutation classes. The first one is the theory living on D3s probing F_0, the zeroth Hirzebruch surface, which is isomorphic to $\mathbb{P}^1 \times \mathbb{P}^1$, as depicted in **Q4**

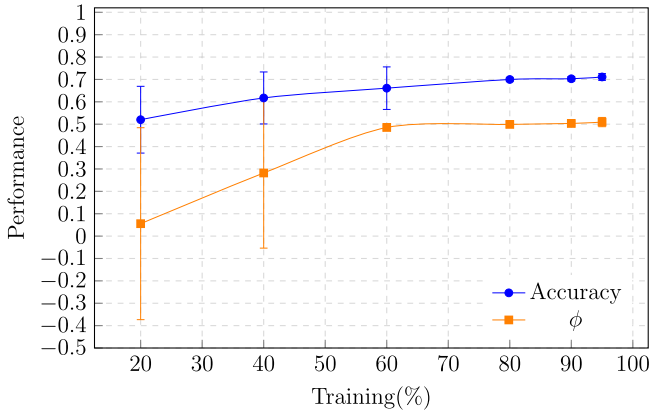


FIG. 3. Training two classes ['A,' 6] and ['D,' 6], and validating ['E,' 6] (0's from ['E,' 6]-['A,' 6] and ['E,' 6]-['D,' 6] pairs). We generate 517, 572, and 600 matrices, respectively. We choose training data out of 14182 pairs and validation data out of 13897 pairs. There are *no* indeterminate ϕ's. The method is NB.

TABLE III. Learning on ADE quivers with the random forest method. Compared to Fig. 3, Naive Bayes performs superiorly and never gives indeterminate results.

Accuracy [%]	60.3214	46.1786	50.2143	51.2857	53.2500
ϕ	0.3468220	−0.2088830	Indeterminate	−0.0184085	0.1678070
Accuracy [%]	54.8571	49.8929	47.5714	49.6071	50.6071
ϕ	0.2026570	−0.0377965	−0.1103220	0.0925057	Indeterminate

[57,58]. The second one is generated by the quiver and adjacency matrix given in **Q5**, which is also anomaly-free.

The learning result of fivefold cross validation is tabulated in Table IV. We also plot the learning curve at different training percentages in Fig. 12, showing results as perfect as the example of ['A,' 4] and ['D,' 4]. It is also worth noting that comparing Fig. 12 with Fig. 11, we see that the learning curve now looks smoother and more beautiful when we use NB.

Now that infinite mutation types generate infinitely many quivers under the Seiberg duality mutation, we can do something that is not done in finite mutations. In the training dataset T, we include the matrices generated to some depth (equal to the number of mutations from the original quiver) in the duality tree. However, the validation dataset V consists of matrices generated at depths that are far away from those in T. We still start with the above two matrices and generate $(102 + 138)$ matrices. From these matrices, we create 6933 1's and 6358 0's. Then the 1's and 0's of T will be evenly chosen out of the 13291 pairs. For V, we start with the following matrices:

$$\begin{pmatrix} 0 & 211 & -16644 & 765262 \\ -211 & 0 & -1658 & 76232 \\ 16644 & 1658 & 0 & -46 \\ -765262 & -76232 & 46 & 0 \end{pmatrix}, \quad \begin{pmatrix} 0 & -2586 & 39 & 55 \\ 2586 & 0 & 39603 & -47 \\ -39 & -39603 & 0 & 843 \\ -55 & 47 & -843 & 0 \end{pmatrix}, \tag{3.2}$$

and generate $(161 + 161)$ matrices. From these matrices, we create 5689 1's and 5663 0's. Then the 1's and 0's of V will be evenly chosen out of the 11352 pairs. We make a dataset with 12000 pairs in all. At 90% training percentage, the result is tabulated in Table V. This shows that the machine is just guessing. Since it is predicting those of unseen depths, the result is not very surprising. As a matter of fact, the confusion matrices always have a vanishing TP (actual=predicted=1) and an extremely small FP (actual = 0, predicted = 1). This shows that the machine tends to regard the pairs from unseen depths as unrelated theories.

TABLE IV. Training and validating two classes: **Q4** and **Q5**. We generate $(102 + 138)$ matrices. There are 6344 1's and 6268 0's. The method is NB.

Accuracy	F-Score	ϕ
1 ± 0	1 ± 0	1 ± 0

We now have seen that the machine does a good job for validation, but does not perform well when meeting unseen depths far away. It would be natural to ask, given both matrices of depth 0 to depth n_1 and of depth n_2 to depth n_3 $(n_3 > n_2 > n_1 > 0)$, whether the machine can extrapolate the matrices of depths between n_1 and n_2. We still contemplate the above case with two different mutation classes (**Q4** and **Q5**), but this time, we have $n_1 = 3$, $n_2 = 6$, and $n_3 = 8$ for both of the two classes (and hence, we are validating matrices of depths 4 and 5).[11] The learning result at 90% training percentage is listed in Table VI. We can see that the result is better than the one in Table V. It is very natural to expect this since we are having many more matrices trained (or more precisely, the ratio of seen against unseen matrices is much larger). On the other hand, we should also expect that the result would still have much room to be improved regarding the feature of NB.

We now make a proposal using the assumption of NB. As discussed in Sec. III B and Appendix A 2, whether a pair of matrices are related to each other by mutations is independent of other matrices. This condition certainly applies here.

We can actually visualize the duality trees of quivers. Examples can be found in Figs. 2 and 7 in [58]. Since a mutation can act on every single node of a quiver, an n-node quiver is *directly* connected to n other dual quivers. This is true for any quiver in any mutation class. Furthermore, the duality tree of an infinite mutation class is apparently infinite. Thus, it does not matter which quiver we choose to start with due to the symmetry of the duality tree.[12] Now, from the example of Table V, we know that the machine is poor at predicting matrices of depths from

[11] In our training set, we also include pairs of 1's from depths 0–3 and depths 6–8. Likewise, in our validation set, we also include pairs of 1's from depths 4–5 and depths 0–3/6–8. Same is for 0's as well.

[12] For a finite mutation, this is also true as the duality tree will finally close and be symmetric.

TABLE V. Training infinite type quiver matrices at low depths from the originals, and validating on matrices at depths far away. The method, NB, performs poorly.

Accuracy [%]	50.0833	51.1667	49.8333	47.8333	47.9167
ϕ	−0.0807034	−0.0922570	−0.0811080	−0.0520199	−0.1067660

TABLE VI. For the training set, we have 19267 1's and 19243 0's. For the validation set, we generate 13020 1's and 13227 0's. Then we choose correspondingly many pairs used for validation, viz. 3946 pairs.

Accuracy [%]	65.3891	65.5761	66.6277	66.2772	65.9500
ϕ	0.333734	0.329995	0.347612	0.333243	0.336945

$(n_1 + 1)$ to n_2 when only matrices within depths n_1 are trained. This can be illustrated as in Fig. 4(a).

Likewise, for the example of Table VI, we have Fig. 4(b). Then we can have a green disk of trained matrices, centered at each point (up to the azimuth) in the blue annulus, tangent to the two boundaries of the blue annulus as shown in Fig. 5(a). We can use such trained green disk/dataset to predict the matrices inside the white annulus bounded by the green disk and the disk of radius n_4. By the same reasoning, the machine would give poor predictions to those matrices. Notice that the disks of radii n_2 and n_4 have a leaf-shaped overlap, which means that given the small blue disk and the green disk as the training set, this leaf would not enjoy a good prediction. If we draw the green disk along the blue annulus, then those green disks, along with the blue disk in the middle, will become the same training set as in Fig. 4(b). The leaf-shaped overlaps will form the white annulus in the middle bounded by the blue disk and the blue annulus, which is the unseen dataset as in Fig. 4(b). Since the machine cannot learn well in the leaf shapes, although the training set is larger [compared to Fig. 4(a)] which may improve the result, as a consequence of mutual independence assumption, the performance of the machine would still not be greatly improved. Nevertheless, we should emphasize that this is mainly due to the particular feature of NB. As we will see in Sec. III E, this illustration for NB would be quite different for NN.

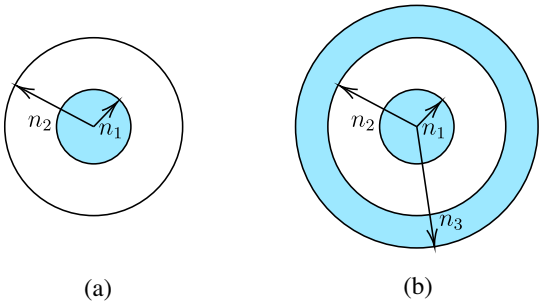


FIG. 4. (a) The blue disk of radius n_1 indicates that the matrices up to depth n_1 are trained. The annulus between circles of radii n_1 and n_2 is the data used in prediction. The behavior of the machine is poor when predicting the matrices in this white annulus. (b) The blue disk and blue annulus indicate the seen matrices in the duality tree. The middle white annulus is used in prediction.

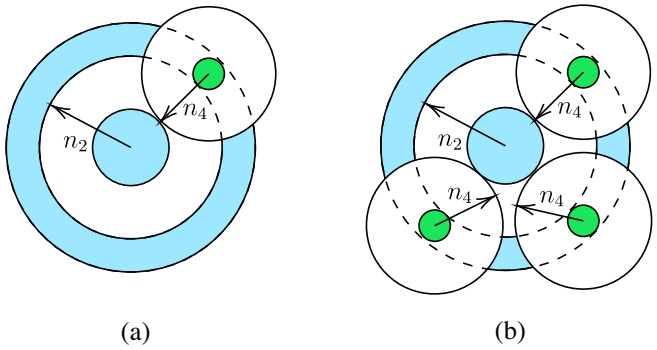


FIG. 5. (a) We can choose a matrix in the blue annulus and generate the green disks. Then the overlap of disks with radii n_2 and n_4 form a leaf shape, whose interior consists of unseen matrices. (b) We can draw all the green disks along the blue annulus. Each disk is contained in a white disk of radius n_4. The green disks then form the blue annulus, and the leaf-shaped overlaps form the big white annulus in the middle.

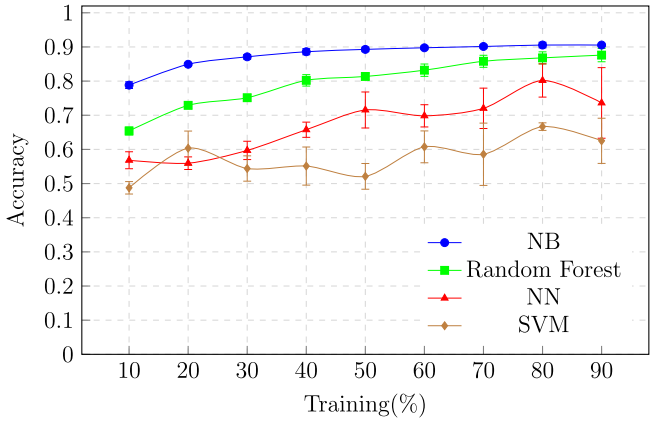


FIG. 6. Training and validating three classes: **Q4**, **Q5**, and **Q6**. We generate $(38 + 48 + 53)$ matrices. There are 2365 1's and 2397 0's.

TABLE VII. Training and validating three classes: ['A,' 6], ['D,' 6], ['E,' 6]. We generate (76 + 77 + 77) matrices. There are 6116 1's and 6049 0's. The method is NB.

Accuracy	F-Score	ϕ
$0.90291800 \pm 0.00920160$	$0.90936100 \pm 0.00886124$	$0.81580000 \pm 0.01625320$

TABLE VIII. Training and validating three classes: **Q4**, **Q5**, and **Q6**. We generate (102 + 138 + 161) matrices. There are 11563 1's and 11482 0's. The method is NB.

Accuracy	F-Score	ϕ
$0.90553300 \pm 0.00970378$	$0.91187400 \pm 0.00831757$	$0.82051800 \pm 0.01696320$

TABLE IX. Training and validating four classes: **Q4**, **Q5**, **Q6**, and **Q7**. We generate (102 + 138 + 161 + 102) matrices. There are 16059 1's and 16250 0's. The method is NB.

Accuracy	F-Score	ϕ
$0.85739200 \pm 0.00750116$	$0.86872800 \pm 0.00563417$	$0.72165300 \pm 0.01548070$

TABLE X. Training and validating five classes: **Q4**, **Q5**, **Q6**, **Q7**, and **Q8**. We generate (102 + 138 + 161 + 102 + 161) matrices. There are 23645 1's and 23698 0's. The method is NB.

Accuracy	F-Score	ϕ
$0.83572900 \pm 0.00292061$	$0.84193700 \pm 0.00320654$	$0.67346000 \pm 0.00515131$

TABLE XI. Training on three infinite type quivers using the neural network method within *Mathematica*'s CLASSIFY function.

Accuracy	F-Score	ϕ
$0.76590900 \pm 0.05281270$	$0.77165100 \pm 0.04618850$	$0.53412100 \pm 0.10287100$

D. Classifying more mutation classes

We now contemplate the datasets containing more mutation classes. It is natural to first consider the case with three mutation classes. We again use ['A,' 6], ['D,' 6], and ['E,' 6] as an example. Of course, unlike the aforementioned case, all three classes have to appear in the training dataset this time. The learning result of fivefold cross validation is reported in Table VII.

The learning curve at different training percentages is given in Fig. 13. We can see that the performance, albeit not as perfect as the cases with two classes, is still very satisfying, with ∼90% accuracies and ∼0.8 Matthews correlation coefficients when only ∼60% of the data is trained.

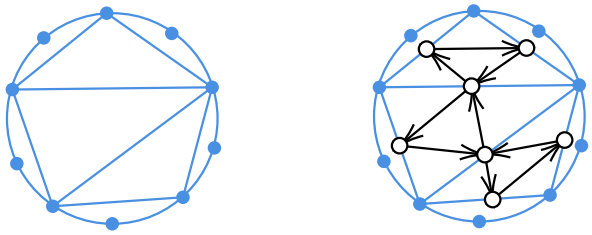
We can also add one more class into the two-class example for **Q4** and **Q5**. The new one is generated by **Q6**. The learning results are reported in Table VIII for fivefold cross validation and Fig. 14 for learning curves. The performance is still very nice, though it is not as perfect as the two-example class.

Let us now contemplate examples with four and five mutation classes. To compare this with the three-class example above, we first choose **Q4**, **Q5**, and **Q6** for our data. For the four-class example, the remaining quiver is depicted in **Q7**.

The learning results are reported in Table IX for fivefold cross validation and Fig. 15 for learning curves.

For the five-class example, we further include **Q8**. The learning results are reported in Table X for fivefold cross validation and Fig. 16 for learning curves. Indeed, we see that the numbers of different classes can affect the performance of the machine.

Nevertheless, a better learning result is always wanted. When we are having more classes, a combinatorial problem arises. If there are more mutation classes in the data, there will be more and more distinct pairs of 0's than pairs of 1's. If we want adequate combinations of 0's, then to keep the dataset well-balanced, correspondingly many 1's are required as well. However, all the distinct pairs of 1's will be included while 0's may still not be enough. On the other



(a) (b)

FIG. 7. The quiver obtained from triangulation of a surface [55]. (a) The triangulation of a 10-gon. (b) The quiver from a triangulated 10-gon. Note that this quiver is in the mutation class of type ['A,' 7].

hand, if we keep adding pairs to our dataset, although we will have more combinations of 0's, there will be duplicated pairs of 1's. These repeated pairs will not be helpful and hence the dataset will be biased. Thus, how the number of mutation classes is (quantitatively) related to the number of matrices generated and the number of pairs assigned is a newly raised question. Roughly speaking, the best way is perhaps to include all the 1's and correspondingly many 0's. Then the number of distinct pairs is maximized while keeping the dataset balanced. Another possible way to resolve this is to use multiclassification with one single matrix as a data point instead of matrix pairs so that the combinatorial problem could be avoided. Let us now contemplate such multiclassifications.

E. Multiclass classifications

For datasets consisting of matrix pairs, we have already seen that NB is the best method for learning mutations. To make this more convincing and clearer, we also plot the learning curves with different methods in Fig. 6 as an example.[13] We also tabulate the fivefold cross validation for NN in Table XI. We should emphasize that the NN here used in *Mathematica* is different from the (C)NN we will use below for multiclassifications. The NN in *Mathematica* CLASSIFY is used for matrix pairs while NN in PYTHON using TensorFlow [59] deals with a single matrix as one data point in the dataset.[14] Unless specified, we will always refer to multiclassifications in PYTHON with TensorFlow when saying NN below.

Besides pairing matrices and assigning 1's and 0's, there is a more direct way to classify theories in distinct duality trees as aforementioned. We can simply assign different mutation classes with different labels, and then let the machine tell which classes the given quivers belong to. So far, we have been using *Mathematica* and its built-in function to do the machine learning. One can still use CLASSIFY and NB to do the training, but it turns out that NB (and the *Mathematica* classifier) is only good when the data are a set of pairs. Thus, we turn to PYTHON to perform machine learning on mutations with the help of SAGE [44] and TensorFlow . Henceforth, when we say that the method is NB (or NN), we simultaneously mean that the type of the dataset used is the one suitable for this method. This time we choose three classes generated by **Q12**, **Q13**, **Q14**. Quiver **Q12** is defined through triangulation of a 10-gon, and this process is shown in Fig. 7.

According to the theorem by Felikson, Shapiro, and Tumarkin [52], the first two classes are finite while the third one is infinite. Now we label the three classes with $[1,0,0]$, $[0,1,0]$, and $[0,0,1]$, respectively. Thus, when the machine predicts $[a_1,a_2,a_3]$, it is giving probabilities of which class the matrix being predicted should belong to, where a_i's are the probabilities of the three classes, respectively. For instance, if the output is $[0.9,0.06,0.04]$, then the machine classifies the matrix into the first class.

We use convolutional neural networks (CNNs) to deal with the dataset which contains $(1547+1956+1828)$ matrices. We find that there is only ~55% of accuracy when 80% of data is trained. However, it is quite remarkable that for the last class, which is the only infinite one, the machine has a 100% accuracy; i.e., it always correctly recognizes the matrices in this class and never misclassifies other matrices to this class. Hence, the machine seems to have learned something related to finite and infinite mutations. We will explore this in Sec. V D.

F. Classifying against random antisymmetric matrices

There is also another possible way to have a machine learning model on quiver mutations. If we are given some quiver and a class of dual theories, we may wonder whether this quiver also belongs to the duals. Therefore, we can train the machine using a specific class of matrices, along with some randomly generated antisymmetric matrices.

So as not to just learn anomalies, when we are dealing with anomaly-free quivers, we should mainly have random matrices that are anomaly-free as well. For simplicity, let us contemplate the 3×3 matrices. As the nullity of a nonzero 3×3 matrix is at most 1, it should be easier to generate matrices that are anomaly-free.[15] We first test the dP_0

[13] At first, we would like to try many more matrices and much larger datasets. However, a normal laptop is not capable of giving the whole learning curve of SVM. Nevertheless, this example with a smaller size can still tell the difference between various methods. Here, although random forest is still inferior to NB, the discrepancy is small. However, one can check that if we include more matrices and more data, the advantage of NB over other methods will be greater.

[14] We should mention that *Mathematica* now also incorporates a complicated neural network, though we are using TensorFlow here for CNN to make a clearer distinction between binary and multiclassifications in our discussions.

[15] For matrices of higher dimensions, anomalous matrices might be more easily generated randomly. What one could do is to use other different known classes of (anomaly-free) quivers to form a randomly generated set.

TABLE XII. Training and validating four classes: ['A', 4], ['D', 4], ['A', (3,1),1], and ['A,' (2,2),1]. We generate $(52 + 50 + 70 + 54)$ matrices. There are 5503 1's and 5512 0's. The method is NB.

Accuracy	F-Score	ϕ
$0.84648200 \pm 0.00502814$	$0.85653500 \pm 0.00533456$	0.69999000 ± 0.0073538

theory, viz. the class generated by **Q9**, with correspondingly many random antisymmetric matrices. We generate matrices up to depth 7, and we have $(382 + 388)$ matrices for training and validation. The learning curves are plotted in Fig. 17.

As we can see, the result is pretty good with ~90% accuracy when only ~60% of data is trained. If we use this model to predict unseen matrices, i.e., the 384 matrices at depth 8 plus 377 random matrices, the prediction can still reach ~97% accuracy. The accuracy for the matrices in the dP_0 duals is ~93% while the accuracy for random matrices is 100%.

IV. EXAMPLES WITH DIFFERENT TYPES

Let us go back to NB with matrix pairs and contemplate a heuristic example with four different classes. We use ['A,' 4], ['D,' 4], ['A,' (3,1),1] and ['A,' (2,2),1] here, where the latter two are called affine types. The learning results are again reported using fivefold cross validation and learning curves as in Table XII and Fig. 18, respectively. Even at 95% training percentage, the accuracy is ~85%, and the Matthews ϕ is only ~0.7. This is certainly not that satisfying.[16]

We can simply put all the finite and affine types we meet so far (['A,' 4], ['D,' 4], ['A,' (3,1),1], ['A,' (2,2),1], ['A,' 6], ['D,' 6], ['E,' 6]) together to create a dataset containing seven different mutation classes. We try the following three experiments:

(1) We generate 52, 50, 70, 54, 76, 77, and 77 matrices, respectively. We have 14821 pairs in our dataset with 7360 1's and 7461 0's.
(2) We generate 144, 50, 120, 54, 76, 77, and 77 matrices, respectively. We have 46332 pairs in our dataset with 22387 1's and 23945 0's.
(3) We generate 144, 50, 120, 54, 200, 213, and 213 matrices, respectively. We have 43588 pairs in our dataset with 21229 1's and 22359 0's.

Notice that in these three experiments, we also have matrix pairs $\{(M_{4\times4}, M_{6\times6}) \to 0\}$ in our data; that is, we also include the trivial zeros from pairs of two quivers with different numbers of nodes. In all of the experiments, when we train 95% of the dataset and validate the remaining 5%, the accuracy is about 70%–80%, and ϕ is about 0.4–0.6. As expected, when we have more mutation classes, the performance of the machine becomes worse.

As a sanity check, we remove $\{(M_{4\times4}, M_{6\times6}) \to 0\}$ in our data. For instance, we generate 52, 50, 70, 54, 76, 77, and 77 matrices, respectively, and create 14254 pairs with 7375 1's and 7529 0's. We find that the accuracy becomes 65%–75%, and ϕ becomes 0.4–0.5. Getting a lower accuracy and a lower ϕ completely makes sense. Quivers with different numbers of nodes are apparently not dual to each other. Henceforth, we will not include pairs of matrices with different dimensions for 0's in our datasets which easily learned to classify as 0's.

A. Dynkin and affine types

So far in this section, we have discussed two different (finite) mutation types. We mainly deal with ADE types and include affine types as well. In light of the above learning results, we wonder whether different types would affect our result. A simple check would involve only two mutation classes with one Dynkin and one affine. For instance, we test ['D,' 4] and ['A,' (3,1),1] here. We pick out two points in the whole learning curve as in Table XIII. The learning result is as perfect as the result in the example of ['A,' 4] and ['D,' 4]. From the viewpoint of machine

TABLE XIII. We machine learn ['D,' 4] and ['A,' (3,1),1] mutation classes. We generate 92 and 104 matrices, respectively. There are 6347 1's and 6320 0's. The method is NB.

	Accuracy [%]				
Training percentage	ϕ				
90%	100	100	100	100	100
	1.00	1.00	1.00	1.00	1.00
55%	100	100	100	100	100
	1.00	1.00	1.00	1.00	1.00

TABLE XIV. We machine learn ['D,' 4]'s and F_0 theory's quiver mutation classes. We generate 92 and 102 matrices, respectively. There are 6313 1's and 6316 0's. The method is NB.

	Accuracy [%]				
Training percentage	ϕ				
90%	100	100	100	100	100
	1.00	1.00	1.00	1.00	1.00
55%	100	100	100	100	100
	1.00	1.00	1.00	1.00	1.00

[16]We already know that the numbers of mutation classes in the training data can affect our result. Nevertheless, it is reasonable to speculate that other factors such as the quiver types may also have influence.

TABLE XV. Training and validating five classes: ['A,' 4], ['D,' 4], ['A,' 6], ['D,' 6] and ['E,' 6]. We generate $(52 + 50 + 76 + 77 + 77)$ matrices. There are 6791 1's and 6726 0's. The method is NB.

Accuracy	F-Score	ϕ
$0.88710300 \pm 0.00751058$	$0.89368900 \pm 0.00711335$	$0.78158800 \pm 0.01641340$

TABLE XVI. Training and validating three classes: ['T,' (4,4,4)], ['T,' (4,5,3)], and ['T,' (4,6,2)]. We generate $(65 + 65 + 66)$ matrices. There are 2553 1's and 2565 0's. The method is NB.

Accuracy	F-Score	ϕ
$0.88569500 \pm 0.00987409$	$0.89199500 \pm 0.00793421$	$0.77648300 \pm 0.01925770$

learning, this is definitely a successful and exciting result. More importantly, our point here is to seek out the influence of different types. We find that learning mutation classes of the same type (e.g., only Dynkin) and learning those of different types (e.g., Dynkin + affine) have the same performance.

Let us further try an example with one finite mutation type (['D,' 4]) and one infinite mutation type. For the infinite one, we choose the quiver **Q4**. We pick out two points on the learning curve as tabulated in Table XIV. We see that it is still as perfect as the case with two Dynkin types (['A,' 4],['D,' 4]). To summarize, the mutation types would not really affect our learning performance for NB.

We return to our example with seven classes (['A,' 4], ['D,' 4], ['A,' (3,1),1], ['A,' (2,2),1], ['A,' 6], ['D,' 6], ['E,' 6]). This time let us remove the two affine types and study the learning performance of the data with five classes. The results are reported in Table XV for fivefold cross validation and 19 for learning curves. We find that the result is improved. It is even better than the result of four classes (['A,' 4], ['D,' 4], ['A,' (3,1),1], ['A,' (2,2),1]). Unlike the above tests, this seems to tell us that the influence from different types outcompetes the influence from the number of mutation classes. However, as we will see next, this is not the real reason.

B. T Type

Now, we perform a test on three infinite classes, all of which are T types [45]: ['T,' (4,4,4)], ['T,' (4,5,3)], and ['T,' (4,6,2)]. A quiver of T type is an orientation of a tree containing a unique trivalent vertex, of three leaves of degree one, and with the remaining vertices in the branches being of degree two. When we say a quiver is of type ['T,' (a, b, c)], we mean there are a total of $(a-2)+(b-2)+(c-2)$ vertices of degree two, summing up the contributions from the three branches. They are all 10×10 matrices.[17] The learning results are given in Table XVI for fivefold cross validation and Fig. 20 for learning curves.

[17]We already know that the sizes of matrices will not have a big influence on our results, so we are free to choose matrices of any dimension.

TABLE XVII. Training and validating four classes: **Q4**, **Q5**, **Q9**, and **Q10**. We generate $(102 + 138 + 94 + 138)$ matrices. There are 12276 1's and 11915 0's. The method is NB.

Accuracy	F-Score	ϕ
1 ± 0	1 ± 0	1 ± 0

We see that the performance is basically the same as the three-infinite-class example in Sec. III D. Therefore we do not see the influence of mutation types here. Again, the influence of numbers of classes should dominate the performance of the CLASSIFY function in *Mathematica*.

C. Splitting the dataset

Let us now try to solve the puzzle left at the end of Sec. IV A. Consider the quivers and matrices in **Q9** and **Q10**. We can machine learn the dataset with these two classes. This yields 100% accuracy and $\phi = 1$ most of the time, which is good as expected. However, we can put these two quivers and the two quivers in Sec. III C (**Q4** and **Q5**) together and machine learn the four classes generated from these four quivers. The fivefold cross validation is given in Table XVII. We also pick out three points on the learning curve, which is tabulated in Table XVIII.

Unlike the usual result one should expect from a four-class case, this learning result is almost as good as two-class cases. In fact, this is the key. Since we have two classes of 3×3 matrices and two classes of 4×4 matrices, the machine actually splits the dataset into two pieces; viz. it treats 3×3 and 4×4 matrices separately. Just as including zeros from pairs of matrices of different sizes, although machine learning is not affected by dimensions of matrices *longitudinally*,[18] there is a *transversal* influence of

[18]For the sake of brevity, by this, we mean that if we have two datasets with, say, k different mutation classes of $m \times m$ matrices and k different mutation classes of $n \times n$ matrices ($m \neq n$), the performance should roughly be the same. On the other hand, if we have matrices of different sizes in one dataset, we shall say that we are studying how the matrix dimensions affect the results transversally.

TABLE XVIII. We generate 102, 138, 94, and 138 matrices, respectively. There are 13199 1's and 12469 0's.

	Accuracy [%]				
Training percentage			ϕ		
90%	100.0000	100.0000	100.0000	100.0000	100.0000
	1.000000	1.000000	1.000000	1.000000	1.000000
80%	100.0000	100.0000	100.0000	100.0000	100.0000
	1.000000	1.000000	1.000000	1.000000	1.000000
50%	99.9532	99.9844	99.9922	99.9844	99.9922
	0.999065	0.999688	0.999844	0.999688	0.999844

the matrix dimensions. Now we are able to explain why in Sec. IV A, the example with five classes (['A,' 4], ['D,' 4], ['A,' 6], ['D,' 6], ['E,' 6]) has a better result than the one with four classes (['A,' 4], ['D,' 4], ['A,' (3,1),1], ['A,' (2,2),1]). Effectively, the machine is dealing with $(2+3)$ classes and 4 classes, respectively.

V. ENHANCING THE DATASET

A. Adding ranks of nodes for NB

Since physically interesting quivers have (round) nodes as gauge groups, each node carries the rank information of the gauge group. Thus, we can further add the rank information to "help" the machine learn Seiberg duality. Above all, these quivers should be anomaly-free, which is encoded by the kernel of the adjacency matrix M with certain rules under Seiberg duality as discussed in Sec. II A [60,61]. We simply add the ranks of nodes as a column vector $\boldsymbol{v}$ to our dataset by

$$\{(M_1, \boldsymbol{v}_1), (M_2, \boldsymbol{v}_2) \to 1/0\}. \tag{5.1}$$

We first test this on three classes as in **Q4**, **Q5**, and **Q6**. The results are given in Table XIX for fivefold cross validation and Fig. 21 for learning curves. We find that the learning result is the same compared to the former example with bare matrix input.

Now we add the class generated by **Q7** to our data. The four-class result is reported in Table XX for fivefold cross validation and Fig. 22 for learning curves.

We also further include **Q8** to construct the five-class example with extra rank information. The result can again be found in Table XXI for fivefold cross validation and Fig. 23 for learning curves.

Again, we learn that the learning results are not improved with the extra vectors. Based on the above results, it is possible that the machine already sees the rank information when we only feed it with bare matrix input (since it is related to the adjacency matrix kernels), therefore it does not require us to give the rank vector explicitly.

Moreover, we can try predicting totally unseen matrices as well. Let us use the three-class example (**Q4**, **Q5**, and **Q6**). We still train $(102+138+161)$ matrices, viz. generate to (and include) depths 4. Then our validation contains matrices

TABLE XIX. Training and validating three classes: **Q4**, **Q5**, and **Q6**. We generate $(102+138+161)$ matrices. There are 11506 1's and 11645 0's. The method is NB. The rank information is included.

Accuracy	F-Score	ϕ
$0.91041400 \pm 0.00306970$	$0.91662600 \pm 0.00340356$	$0.82855000 \pm 0.00626524$

TABLE XX. Training and validating four classes: **Q4**, **Q5**, **Q6**, and **Q7**. We generate $(102+138+161+102)$ matrices. There are 13930 1's and 14005 0's. The method is NB. The rank information is included.

Accuracy	F-Score	ϕ
$0.85520000 \pm 0.00674474$	$0.86390500 \pm 0.00619574$	$0.71583900 \pm 0.01142870$

TABLE XXI. Training and validating five classes: **Q4**, **Q5**, **Q6**, **Q7**, and **Q8**. We generate $(102+138+161+102+161)$ matrices. There are 22770 1's and 22823 0's. The method is NB. The rank information is included.

Accuracy	F-Score	ϕ
$0.84267400 \pm 0.00915047$	$0.84683800 \pm 0.00846313$	$0.68633100 \pm 0.01791090$

TABLE XXII. Training and validating three classes: **Q9**, **Q10**, and **Q11**. We generate (94 + 138 + 123) matrices. There are 11271 1's and 11301 0's. The method is NB. The Diophantine variables are included.

Accuracy	F-Score	ϕ
$0.91148800 \pm 0.00091432$	$0.91759000 \pm 0.00115878$	$0.83179900 \pm 0.00135928$

TABLE XXIII. Training and validating three classes: **Q9**, **Q10**, and **Q11**. We generate (94 + 138 + 123) matrices without the augmented Diophantine variable information. There are 11239 1's and 11298 0's. The method is NB. The dataset is composed of bare matrix pairs only.

Accuracy	F-Score	ϕ
$0.91431900 \pm 0.00644304$	$0.91987000 \pm 0.00657059$	$0.83621300 \pm 0.01123010$

of depths 5 and 6, which has (688 + 978 + 1258) matrices. The training set has 12938 1's and 12961 0's while the validation set has 8987 1's and 8974 0's. After picking out correspondingly many pairs from each set, at 90% training, we find that the accuracy is $0.50632400 \pm 0.00932148$, and ϕ is $0.01286830 \pm 0.01174640$. As a result, the performance is the same as before. Therefore, we would say for NB, the machine already sees the rank information to some extent even if we only have bare matrix input.[19]

B. Adding Diophantine variables

It is also natural to ask what would happen if we use some other ways of dataset enhancement. For superconformal chiral quivers, physical constraints should be imposed to those block quivers. The following conditions—chiral anomaly cancellation for the gauge groups, vanishing Novikov-Shifman-Vainshtein-Zakharov (NSVZ) β function for each coupling as well as their weighted sum, and marginality of chiral operators in the superpotential at interacting fixed point—lead to a Diophantine equation [57,61,62].[20] For three-block quivers, the Diophantine equation reads

$$\frac{a_{23}^2}{\alpha_1}+\frac{a_{31}^2}{\alpha_2}+\frac{a_{12}^2}{\alpha_3}=a_{12}a_{23}a_{31}, \tag{5.2}$$

where a_{ij}'s are the numbers of arrows among blocks (i.e., entries of the matrix) and α_i's the numbers of nodes in the blocks. Motivated by this intrinsic structure of the mutation classes rooted in these physical constraints, we simply arrange a_{ij}^2's and $a_{12}a_{23}a_{31}$ (which we shall call Diophantine variables for simplicity) into a vector and add it to the data. Now each pair looks like

$$\{(M,(a_{12}^2,a_{23}^2,a_{31}^2,a_{12}a_{23}a_{31})^{\mathrm{T}}),\\ (N,(b_{12}^2,b_{23}^2,b_{31}^2,b_{12}b_{23}b_{31})^{\mathrm{T}})\to 1/0\}. \tag{5.3}$$

However, we should emphasize that we are not actually telling the machine that the quivers/matrices should obey the Diophantine equation. Otherwise, for instance, for superconformal three-*block* quivers, we would only have 16 of them [60]. We are just using some specific combinations of a_{ij}'s (inspired by Diophantine equations) and putting this extra explicit vector in the data to see if this would give any improvement.

We first try an example with three mutation classes of 3×3 matrices.[21] We use the quivers **Q9**, **Q10**, and **Q11**. We list the fivefold cross validation result in Table XXII. For reference, the learning result without including any extra information/vectors is also given in Table XXIII. We can see that there is no improvement.

Let us now try 4×4 matrices. Again we have three classes as in **Q4**, **Q5**, and **Q6**. The Diophantine equation for four-block quivers reads [61]

$$\begin{aligned}&a_{12}a_{23}a_{34}a_{14}\\&=\frac{a_{12}^2}{\alpha_3\alpha_4}+\frac{a_{13}^2}{\alpha_2\alpha_4}+\frac{a_{14}^2}{\alpha_2\alpha_3}+\frac{a_{23}^2}{\alpha_1\alpha_4}+\frac{a_{24}^2}{\alpha_1\alpha_3}+\frac{a_{34}^2}{\alpha_1\alpha_2}\\&\quad+\frac{a_{12}a_{24}a_{14}}{\alpha_3}-\frac{a_{12}a_{23}a_{13}}{\alpha_4}+\frac{a_{13}a_{34}a_{13}}{\alpha_4}-\frac{a_{23}a_{34}a_{24}}{\alpha_1}.\end{aligned} \tag{5.4}$$

We therefore add the vector

$$(a_{12}^2,a_{13}^2,a_{14}^2,a_{23}^2,a_{24}^2,a_{34}^2,a_{12}a_{24}a_{14},a_{12}a_{23}a_{13},\\ a_{13}a_{34}a_{14},a_{23}a_{34}a_{24},a_{12}a_{23}a_{34}a_{14})^{\mathrm{T}} \tag{5.5}$$

[19]However, as we will see shortly, rank information would make improvements when we have neural network and use multiclassification.

[20]More generally, monodromies give rise to mutation invariants, which in turn can be formulated as a set of Diophantine equations characterizing the space of dual theories (see, e.g., [6,63]).

[21]Since we have already seen that the machine almost always gives correct predictions for two classes, we will start from three classes.

TABLE XXIV. Training and validating three classes: **Q4**, **Q5**, and **Q6**. We generate (102 + 138 + 161) matrices. There are 11490 1's and 11449 0's. The method is NB. The Diophantine variables are included.

Accuracy	F-Score	ϕ
0.90980400 ± 0.00358550	0.91565100 ± 0.00323882	0.82811200 ± 0.0057953

TABLE XXV. Training and validating four classes: **Q4**, **Q5**, **Q6**, and **Q7**. We generate (102 + 138 + 161 + 102) matrices. There are 14040 1's and 14109 0's. The method is NB. The Diophantine variables are included.

Accuracy	F-Score	ϕ
0.858965 ± 0.00349098	0.868007 ± 0.0032153	0.72425500 ± 0.00712124

TABLE XXVI. Training and validating five classes: **Q4**, **Q5**, **Q6**, **Q7**, and **Q8**. We generate (102 + 138 + 161 + 102 + 161) matrices. There are 23211 1's and 23316 0's. The method is NB. The Diophantine variables are included.

Accuracy	F-Score	ϕ
0.84443400 ± 0.00325140	0.84887800 ± 0.00285711	0.68986700 ± 0.00652878

to our data.[22] The learning results are given in Table XXIV and Fig. 24 for fivefold cross validation. The performance is not really improved.

Let us contemplate an example with four mutation classes. This time, we use the quivers **Q4**, **Q5**, **Q6**, and **Q7**. We report the results in Table XXV for fivefold cross validation and Fig. 25 for learning curves. Again, the performance is the same.

Now move on to the case with five mutation classes. Besides the above four matrices, we further include the quiver **Q8**. The experiment without adding the Diophantine variables is done in Sec. III D. The new learning results are given in Table XXVI for fivefold cross validation and Fig. 26 for learning curves. We find that this is still not improved.

Moreover, we can try predicting totally unseen matrices as well. Let us use the three-class example (**Q4**, **Q5**, and **Q6**). We still train (102 + 138 + 161) matrices, viz. generate to (and include) depths 4. Then our validation contains matrices of depths 5 and 6, which has (688 + 978 + 1258) matrices. The training set has 12886 1's and 13029 0's while the validation set has 8979 1's and 8981 0's. After picking out correspondingly many pairs from each set, at 90% training, we find that the accuracy is 0.50191000 ± 0.01061240, and ϕ is 0.00206997 ± 0.025543800. We also have the similar experiment for NN, where this extra Diophantine-inspired structure does not improve learning as well. This suggests that such information does not help encode the structure of the quivers, which may be reasonable as we are also considering more general quivers and classes.

[22]Again, we are essentially adding these specific combinations of variables to the dataset, not the equation.

C. Adding ranks of nodes for NN

Now back to the example of **Q12**, **Q13**, and **Q14** in the multiclass classification, let us add the rank information to our dataset by augmenting the data input matrices to include the rank vectors as before. We have (496 + 898 + 484) matrices for training and validation. The learning curves of accuracies are plotted in Fig. 8. We can see that the result is greatly improved after we include the rank information. With enough data trained, the accuracies approach 1, which is much better than the examples using NB. We also notice that at a very low training percentage, the machine again confuses the two finite mutation classes while it almost always gives correct results for the infinite one.[23] The test without rank information above looks like the "limit" at a low training percentage of the test with rank information. To see whether this model is really useful, we use it to predict matrices at unseen depths in these classes. For the predicted (1051 + 3263 + 1344) matrices, we get ∼74% accuracy and ∼71% F1 score. Although this has not reached perfectness, in particular for the purpose of application, the result for unseen matrices is still much better than those in NB. It is not just guessing any more, and we are on track to further improve this.

D. Finite and infinite mutations

Recall that in Sec. III E, the machines seem to treat finite and infinite mutations separately. Hence, we replace the

[23]Notice that the machine tends to classify the matrices in the first class as in the second class when making mistakes. This is due to the imbalance in the data. In spite of this, we can still get a very good result.

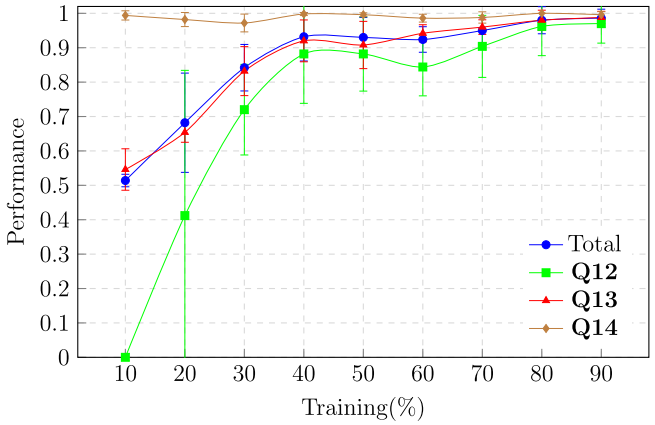


FIG. 8. Training and validating three classes: **Q12**, **Q13**, and **Q14**. We have $(496 + 898 + 484)$ matrices. We use multiclass classification in NN. The rank information is included via imposing the null vector. The learning curves are all accuracies.

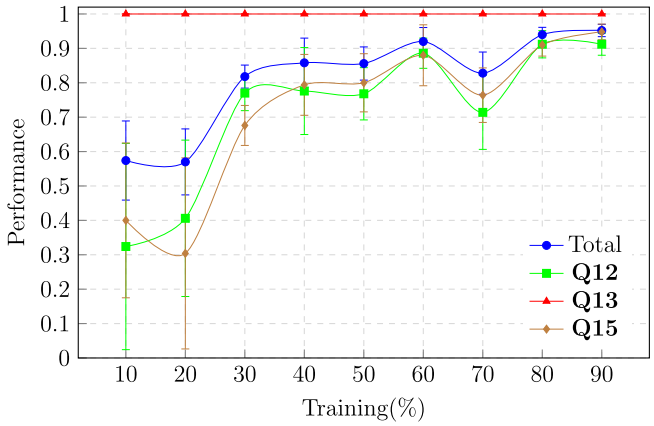


FIG. 9. Training and validating three classes: **Q12**, **Q13**, and **Q15**. We have $(496 + 499 + 484)$ matrices. We use multiclass classification in NN. The rank information is included via imposing the null vector. The learning curves are all accuracies.

infinite one (**Q14**) with another finite class as shown in **Q15**, which is anomalous.

We have tried CNN, as well as MLP and RNN, and find that all of them predict $[\sim 0.333, \sim 0.333, \sim 0.333]$. This means that the machine is not able to decide the classes of the matrices. Hence, comparing the two examples (**Q11-13** and **Q11,12,14**), whether a mutation class is finite or infinite could affect the learning result. More precisely, the machine is learning something that helps it distinguish between finite and infinite mutation types.

We can also include the rank information for the example of **Q12**, **Q13**, and **Q15**. Although the quiver **Q15** is anomalous, we can still assign some vector, say $(1, 1, 1, 1, 1, 1)^{\mathrm{T}}$ to it. Then the anomalies for every node should still add some consistent information on the duality operation among duals.[24] We have $(496 + 484 + 499)$ matrices for training and validation,[25] and the model will be used to predict $(1051 + 1344 + 1631)$ matrices. For training and validation, the learning curves are plotted in Fig. 9. We can see that with enough training, the result is still very good. It is also worth noting that when the machine meets a matrix belonging to the second class (**Q13**), it never misclassifies the matrix to other classes; viz. the red learning curve is a constant equal to 100%. Now for prediction, the machine again gives ∼71% accuracy and ∼0.71 F1 score.

The above two examples show promising results for both physicists and mathematicians. We see that imposing rank information in NN significantly improves the performance of the machine to learn Seiberg duality. From a pure mathematical point of view, in particular the second example with all finite mutation types, this shows that the machine can learn which quivers are from which surfaces (or the 11 sporadic quivers) if we enhance the data as above.

E. Predicting matrices at middle depths

Now we would like to know whether the results for unseen data in predictions can be improved. Our strategy is again to train the matrices up to some depths, as well as some matrices at depths far away. Then we can check how NN behaves when predicting the matrices at middle depths. As a toy model, we train the matrices generated from **Q12**, **Q13**, and **Q14** at depths 0–3 and 5. Then we use the trained model to predict the $(351 + 705 + 350)$ matrices at depth 4. In order to have a more balanced dataset, we choose 1062 matrices out of 3263 matrices at depth 5 for the class of **Q13**. Therefore, we have $(1196 + 1255 + 1478)$ matrices for training and validation. We train 90% and validate the remaining 10% for our model, which gives almost always 100% accuracy as expected. Impressively, after repeating training/validation and prediction a few times, we find that the machine almost always gives 100% accuracy on the matrices at unseen depth (with only several errors out of tens of thousands of predictions, and in particular these few errors never happen for the infinite class). Such things do not happen for the NB cases. This is a perfect result, especially in the sense of application of machine learning on quiver mutations. It means that we can have a model to make good predictions on data of a different style to the training data (here at unseen depths).

One may also wonder whether things would change if more mutation classes are involved. Hence, we further include **Q15** to the above dataset. For just training and validation, we find that the result is still that good. Having more classes does not seem to affect the learning result too much. Now we apply this model to matrices at unseen depth just like the above case. Again, the machine gives ∼98% accuracy and ∼0.98 F1 score, which is an impressive result.

[24]Incidentally, this is also true for anomaly-free quivers. For example, the rank of **Q14** is $(1, 1, 2, 1, 1, 1, 1)^{\mathrm{T}}$, but we can get the same good result if we assign a different vector, say $(1, 1, 1, 1, 1, 1, 1)^{\mathrm{T}}$, as long as the following generated quivers and additional vectors are consistent with this choice.

[25]This time we do not choose all the 614 matrices in 0–4 depths for the third class so that the data would not be biased.

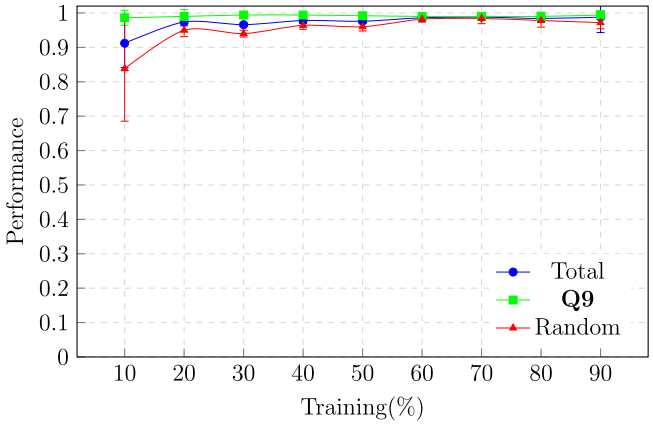


FIG. 10. Training and validating one class, **Q9**, with random matrices. We have (382 + 384) matrices. We use classification in NN. The rank information is included via imposing the null vectors. The learning curves are all accuracies.

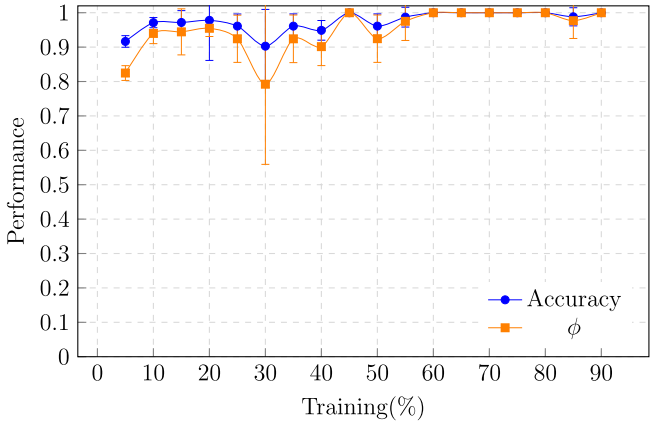


FIG. 11. Training and validating two classes: ['A,' 4] and ['D,' 4]. We generate (144 + 50) matrices. There are 11784 1's and 7200 0's. The method is automatically chosen by the machine within *Mathematica*'s CLASSIFY function.

F. Classifying against random antisymmetric matrices

Let us do the same test involving randomly generated antisymmetric matrices again, but with rank information included. We still generate the matrices to depth 7 so that there are 382 matrices. We train these together with 384 random antisymmetric matrices. The learning curves are plotted in Fig. 10.[26] As we can see, this again improves the result significantly. Even at a low training percentage, the accuracy still looks perfect. Now we use this to predict the 384 matrices at depth 8, along with 461 unseen random matrices. It turns out the accuracy is almost 100%, with roughly ten mistakes only. Thus, if we would like to know whether a quiver belongs to some specific class of theories, this kind of model would be very useful. It is also worth noting that here we do not even need to include matrices at depths outside those used for predictions.

We can further try an example with two classes and some random matrices. This time, **Q10** is involved as well. We now generate to depth 6 and choose 384 out of the 506 matrices for this newly added class. It turns out at 90% training, the accuracy is only 0.8460000 ± 0.0336155, with the F1 score being 0.8420000 ± 0.0258844. If we use this model to predict matrices at the next unseen depths, along with unseen random matrices, the accuracy is ~80%, with the F1 score being ~0.81. This does not decrease too much compared to the validation result. However, using a NN to identify whether a random quiver belongs to a particular duality class works best when considering only one class at a time.

[26]Incidentally, one can still try to use CLASSIFY and NB in *Mathematica*. However, as aforementioned, NB is only good when the data are a set of pairs. For the example here, even at 90% training, the accuracy is only 0.4619350 ± 0.0148527. Even if we try only two classes (without random matrices), but not making pairs, the accuracy is only 0.6835440 ± 0.2462260.

VI. CONCLUSIONS AND OUTLOOK

Based on all the tests above, we can see that Seiberg duality and quiver mutations are very machine learnable. Several points are summarized as below. We first list the conclusions for NB and *Mathematica* classifier:

(i) The number of different mutation classes is the *dominant* influence in our machine learning. Fewer classes in the dataset would give better learning results. Other factors (such as mutation types, dimensions of matrices, and adding rank information) are outcompeted for influence on the learning when there is a larger number of mutation classes.

(ii) One reason that numbers of classes greatly affect our result would be the large number of matrices we have. In particular, (# [combinations of assigning 0]− #[combinations of assigning 1]) gets larger when we include more mutation classes. We need to find a balance between avoiding duplicated 1's and taking care of various combinations of 0's. Our strategy would be to generate as many distinct 1's as possible, and then to generate approximately the same number of 0's. Thus, we could maximize the combinations of 0 without duplicated 1's while keeping the dataset unbiased.

(iii) The dimensions of matrices affect the result "transversally" rather than "longitudinally." If we have two datasets with, say, k different mutation classes of $m \times m$ matrices and k' different mutation classes of $n \times n$ matrices ($m \neq n$), the performance should be roughly the same. On the other hand, the machine would spontaneously split the data into smaller parts in terms of the dimensions of matrices. For instance, a dataset with two classes of 4×4 matrices and three classes of 5×5 matrices would lead to a better result than the dataset four classes of 4×4 matrices does. The former effectively has $(2 + 3)$ classes, and

hence the machine would have better performance in contrast to those with pure four or five classes. Of course, the $(2+3)$-class case would still be a bit worse than a pure two-class example. Moreover, in light of the above two points, we shall *never* include trivial 0's where each pair consists of matrices with different sizes. Although the transversal influence of dimensions does improve our result, this would bring a larger discrepancy between combinations of 0's and 1's, which can be cumbersome as aforementioned, especially for the dataset with many mutation classes. Now that these 0's represent theories that are obviously not dual to each other, there is no necessity to have them in the dataset.

(iv) NB is the best method in the CLASSIFY function due to its mutual independence assumption.

(v) The NB classifier already sees the hint of rank information when we have only bare matrices as input, and thus imposing rank information would not further improve the machine learning result of the NB classifier.

(vi) When the machine encounters mutation classes that are not seen in the training data, the performance gets worse. This is a reasonable result.

For multiclass classifications (and cases with random antisymmetric matrices), we mainly use CNNs here, and we see they behave differently compared to NB. What NB is good at does not seem to work for a NN method, and vice versa. NB gives good results when the data are arranged in pairs while NN has great performance in multiclass classifications. It turns out that NN would be more useful in the application of machine learning mutations in light of the following points:

(i) We find that NN can distinguish whether a mutation class is finite or infinite, even without adding rank information. If we have a finite (infinite) mutation class among infinite (finite) mutation classes, the machine can almost always give 100% accuracy to single out that finite/infinite class.

(ii) We can impose the ranks as additional vectors augmented to the matrices. Then an NN classifier can give extremely good results for validation. This means the ranks of nodes would somehow reveal the structure behind a quiver to some extent. If we include some matrices at depths far away, then the unseen matrices at middle depths can be perfectly classified [as depicted in Fig. 4(b)].[27] The machine almost always give nearly 100% accuracy when making predictions. Furthermore, the number of distinct mutation classes does not seem to strongly affect the performance of NN in this case.

(iii) We can train one class of matrices with some other randomly generated matrices. Even without rank information, the results are still quite nice (e.g., see the results at the end of Sec. III F). To improve these results, including rank information can bring great improvements. If we use this model to predict matrices at unseen depths in that class [as depicted in Fig. 4(a)], as well as unseen random matrices, the results are still almost-perfect (i.e., almost 100% accuracy). Unlike the above bullet point, this does not even require matrices at depths far away to be involved in training. However, this kind of model only works best for classification with one class (against the random matrices). Having more classes would make it lose efficacy (e.g., two classes plus random matrices would decrease the accuracy of predictions to 80%).

We see that ∼100% accuracy for *predictions* can be obtained in all the above three points. These are the key results that might be useful in real-world application.

Outlook It would also be interesting to ask whether the machine can recognize totally unseen *classes* (rather than just matrices at unseen depths in trained classes) after training. For NB and *Mathematica*, we can use matrix pairs and the predictions on pairs involving unseen classes will still be 0 or 1. However, as we have already seen, such a model is poor at prediction on unseen data; hence it may not be that useful here. On the other hand, NN performs well for predictions. However, it is not suitable for a dataset with matrix pairs. Therefore, we can apply these classification networks to only multiclass classification problems. Unfortunately, due to the problem structure of multiclassification, NNs can only recognize, and classify into, categories that are trained. When meeting an unseen class, it would treat the matrix as some element from a trained class. The design of supervised learning used with these NNs implies no machine can even tell that such a matrix does not belong to any trained class, let alone recognize a totally unseen class. Perhaps the closest realization so far would be the model containing random matrices. Then the machine would at least know that the unseen classes are different from the class being trained.

Thus, it would be natural to ask whether the advantages of the above two methods can be combined. NB has better behavior when the matrices are paired, and NN can have really good results when dealing with matrices at unseen depths. From the perspective of machine learning, the network structure, such as the choices of layers and loss functions, might be improved. We hope that in the future we can develop new techniques for our models, especially for NNs or similar models, to make good predictions for matrix pairs and hence be useful for unseen classes.

More generally, we can imagine training the machine with a large number of pairs consisting of a randomly generated quiver and a dual connected to it by a single Seiberg duality on one of its nodes. We could then

[27]Notice the argument on unseen matrices when discussing NB does not apply here for NN, as we have already seen from the learning results. This should be due to NB's mutual independence condition, while NN does not have this.

investigate if the machine can determine whether a pair of quivers are dual. If successful, this would arguably amount to the machine "learning Seiberg duality."

There are many other directions for future work as well. For instance, supervised learning is used in this paper. We would also like to see what would happen if we do not label the matrices and let the machine learn without supervision. We are also not taking superpotentials into account here. All the bidirectional arrows get canceled as we integrate out these fields. It would be intriguing to explore nontrivial superpotential quivers. Such data may be constructed with the help of Kasteleyn matrices [8,9]. Moreover, similar to what we have done for Seiberg duality in $4d$, we can try applying the machine to $2d$ $\mathcal{N}=(0,2)$ triality [64,65], to $0d$ $\mathcal{N}=1$ quadrality [66], and to the order $(m+1)$ dualities of m-graded quivers that generalize them [17]. It is also worth noting that in [22], machine learning is applied to D-branes probing toric Calabi-Yau (CY) cones. Therefore, it is possible for us to study volume minimizations with machine learning. Finally, it would be interesting to ask whether the concept of finite types could be machine learned. Such types are exactly the ADE Dynkin types and their matrices have eigenvalues less than 2 [67]. Matrices and their eigenspaces are ubiquitous in mathematics, physics, and machine learning. This would lead to a deeper study of matrices in machine learning.

ACKNOWLEDGMENTS

The authors thank the Institute for Mathematics and its Applications for their hospitality and for their hosting of the workshop "SageMath and Macaulay2: An Open Source Initiative" that inspired the genesis of this paper. The open source software SAGE [44], including its cluster algebra and quiver package [45], was especially fundamental to this project. J. B. thanks Zijing Wu for useful discussions. The research of S. F. was supported by the U.S. National Science Foundation Grants No. PHY-1820721 and No. DMS-1854179. Y. H. H. thanks STFC for Grant No. ST/J00037X/1. E. H. thanks STFC for the Ph.D. studentship. G. M. thanks the NSF for Grants No. DMS-1745638 and No. 1854162.

APPENDIX A: MACHINE LEARNING STRUCTURE

1. *Mathematica*'s CLASSIFY

Within the *Mathematica* software, the CLASSIFY function allows analysis of a variety of allowed input data types. These input data types include strings, sounds, and images, as well as the familiar numerical inputs. In our case the input data are tensor structures with integer entries. It may hence be noted that the generality of this function's data inputs may reduce the likelihood of it being optimized for use exclusively with tensors.

The CLASSIFY function takes as input training and validation sets, and in our case these were lists of pairs of square matrices (or pairs of matrices along with vectors of their respective rank data). In addition, within the calling of the function, the user can specify the classification method used, as well as the classification performance goal, and even allow the option for the pseudorandom number seeding for the classification process.

The performance goal used was the standard "automatic" option. This selection calculates a weighted tradeoff for the final classifier that is trained such that it has high accuracy of output while still running quickly in subsequent classifications, and not requiring excessive memory storage.

More importantly in the creation of the classifier is the classification method used. *Mathematica* allows nine method options, which among them include Decision Trees, Markov Sequence Classifiers, Support Vector Machines, and Simple Artificial Neural Networks. When running CLASSIFY without specifying a method, the program will run all methods and output a learning curve to allow comparison of performance between the methods on the input dataset (using parameters for comparison based on the validation data) [68].

In initial testing of the CLASSIFY function with some of the datasets, the Naive Bayes method was consistently superior in the performance of its classifier. This is linked to the independence of the pair structure of the input data. Therefore, to avoid superfluous classifier training the method was specified to be Naive Bayes for the remainder of the investigation. Further discussion of the design and success of this method is discussed in Appendix A 2.

2. The Naive Bayes method

We have seen that the Naive Bayes method, as a machine learning classifier, always gives us the best result when applying the built-in CLASSIFY to learn the matrix mutations. Essentially, our model is a conditional probability problem: $p(v_i|T)$, where T acts as the condition for the machine to predict each $v_i \in V$ to be 0 or 1. Then Bayes' theorem yields

$$p(v_i|T) = \frac{p(T|v_i)p(v_i)}{p(T)} = \frac{p(T,v_i)}{p(T)}. \tag{A1}$$

Since $p(T)$ does not affect our result as this is solely determined by the fixed training set $T=\{t_1,t_2,\ldots,t_n\}$ in each single experiment, we can fixate on the numerator:

$$\begin{aligned} &p(T,v_i) \\ &= p(t_1,\ldots,t_n,v_i) \\ &= p(t_1|t_2,\ldots,t_n,v_i)p(t_2,\ldots,t_n,v_i) \\ &= \cdots \\ &= p(t_1|t_2,\ldots,t_n,v_i)p(t_2|t_3,\ldots,t_n,v_i)\cdots p(t_n|v_i)p(v_i). \end{aligned} \tag{A2}$$

Naive Bayes is "naive" because it assumes that every t_i is *independent* of the other conditions in T, which is exactly

the property of matrix mutations. Whether a pair of matrices/quivers are related by mutations is always *independent* of other matrices/quivers. This is the reason why the NB method is always the ideal choice.

Therefore, we may omit all the t_k's in the conditional probability of t_j, viz.

$$p(t_j|t_{j+1},\ldots,t_n,v_i) = p(t_j|v_i). \tag{A3}$$

As a result, we have

$$p(v_i|T) \propto p(v_i)\prod_j p(t_j|v_i). \tag{A4}$$

For our binary classification, the output is either 0 or 1. Then the Bayesian classifier C_{B} should output n ($n = 0, 1$) if $p(v_i = n|T) \geq p(v_i = 1-n|T)$ [69]. Hence, we require

$$C_{\mathrm{B}}(v_i = n) = \frac{p(v_i = n|T)}{p(v_i = 1-n|T)} \geq 1. \tag{A5}$$

For the NB classifier, we get

$$C_{\mathrm{NB}}(v_i = n) = \frac{p(v_i = n)}{p(v_i = 1-n)}\prod_j \frac{p(t_j|v_i = n)}{p(t_j|v_i = 1-n)} \geq 1. \tag{A6}$$

As NB is the simplest (Bayes) network, it is often faster than other methods. More importantly, the assumption of conditional independence in NB reflects the special feature of the data.

3. PYTHON's CNNs

In investigations requiring multiclass classification, a more technical machine learning structure is needed to allow high-performance classification. To facilitate this the TensorFlow library and within this the machine learning specific sublibrary KERAS were used [59].

Artificial NNs are code structures for nonlinear function fitting. Their design was generally inspired by that of a biological brain, and they have seen significant success in recent years where computation speed can now account for the computational inefficiency of using these networks compared to traditional algorithms. The networks used in this investigation were dense and deep, in that they had all neurons fully connected between layers, and there were multiple hidden layers in the network.

More specifically the network style used was a CNN. The defining feature of these networks is the local action at the neurons in the hidden layers which preserves the multidimensional structure of the tensor input, acting with a simple linear $2d$ function, and then applying nonlinear activation. Important to stress is the importance of the nonlinearity in the activation functions at each neuron, allowing NNs to well address problems of higher complexity. These networks are traditionally used for image recognition, as the use of convolution is good for identifying local structure in arrays with dimension larger than 1—this motivated their use for this matrix-based datatype [70].

The specific CNN used in this investigation had a sequential structure such that it was a linear stack of layers. The network had three convolutional layers, each with LeakyReLU activation, and each followed by a Maxpooling layer. Then there are two generic dense layers, one with LeakyReLU activation and the other with softmax activation. The Maxpooling layers simply assign to an entry the maximum value of a set of some of the surrounding entries. They are traditionally used in the CNN structure.

LeakyReLU was used as the standard activation function at each layer. This activation is simple to compute, it is monotonic, and it is inherently nonlinear, with the added benefit of fast gradient descent in training due to its proportional derivative form. This function leaves positive inputs to the neuron unchanged, but scales negative inputs down (in our case by a factor of 10). The additional dense layers are needed in CNNs to recreate the vector data structure for classification. Softmax was used as the final activation as it is a sigmoid equivalent, however with traditionally better results and a normalized output essential for classification problems with multiple classes.

When compiling the NN, additional inputs of loss function, optimizer, and metric are required. The loss function is a measure of the performance of the model, it is the function whose optimal value will indicate a well-trained NN, and hence it is a good model. "Mean squared error" was used for the loss function in this investigation, this measure is simple, and it is computationally inexpensive. It is calculated as the sum of squares of the difference between each input and its predicted value by the model, therefore the output values used in training are vector floats bounded by 0 and 1 to reflect the hot encoding of the Boolean output nature in this classification. The optimizer is the method by which the parameters of the network are updated in accordance with the performance of the loss function. Here the "Adam" optimizer was used, which is an inexpensive first-order gradient based method [71]. Finally, the metric used was "accuracy," and this gives the final measure of the NNs' performance and is simply the proportion of correct classifications the model performs on the validation dataset.

4. Measures of the machine's performance

Measures of the performance of a classification method are essential for justifying the use of machine learning. The most standard measure of a classifier is "accuracy." As mentioned in Appendix A 3, this is the proportion of correct classifications performed by the classifier on a validation dataset. To ensure the measure is unbiased, it is important the validation dataset is not used for training while still being representative.

To ensure representative validation datasets, as well as provide a means of calculating error for these measures, k-fold cross validation was used. In these investigations $k = 5$, and hence in each investigation the full dataset (all data points with their respective classification labels) were first randomized, then split into five equal size subdatasets. The machine learning process for training and then validating the classifier was then iterated 5 times, where in each case the validation dataset was a different subdataset from the split, and the training dataset was the remaining four sets combined. For each of the five iterations the measures of performance were calculated and recorded, giving a small dataset for each measure from which mean and standard errors could be calculated [72].

More technical measures of performance used include Matthew's correlation coefficient (MCC, ϕ) and F1 score (also called just F-Score). Both these measures take into account Type I and II errors from misclassification. A Type I error is a "false positive" (FP), where for example a random matrix is classified as in the mutation class, and conversely a Type II error is a "false negative" (FN), where a quiver matrix is classified as not in the class being trained by the machine.

The F1 score measure gives equal weight to Type I and II errors, whereas the MCC measure uses variable weights based on the occurrence of true positives and negatives (TP/TN). These factors make MCC a more favorable measure in this style of binary classification problem [73].

All three measures can be summarized as functions over the "confusion matrix" defined,

$$M = \begin{pmatrix} \mathrm{TP} & \mathrm{FN} \\ \mathrm{FP} & \mathrm{TN} \end{pmatrix}, \tag{A7}$$

such that

$$\begin{aligned} \text{accuracy} &:= \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN}}, \\ \text{F1 score} &:= \frac{2\cdot\mathrm{TP}}{2\cdot\mathrm{TP}+\mathrm{FP}+\mathrm{FN}}, \\ \mathrm{MCC} &:= \frac{\mathrm{TP}\cdot\mathrm{TN}-\mathrm{FP}\cdot\mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})\cdot(\mathrm{TP}+\mathrm{FN})\cdot(\mathrm{TN}+\mathrm{FP})\cdot(\mathrm{TN}+\mathrm{FN})}}. \end{aligned} \tag{A8}$$

The first two measures, accuracy and F1 score, evaluate in the range [0, 1], while the MCC measure takes values in $[-1, 1]$. In all cases a value of 1 indicates perfect prediction of the model. All measures can be generalized to the multiclassification cases also, evaluating in the same ranges.

APPENDIX B: INVESTIGATION LEARNING CURVES

This Appendix section presents additional learning curves calculated for the investigations, as discussed in the paper.

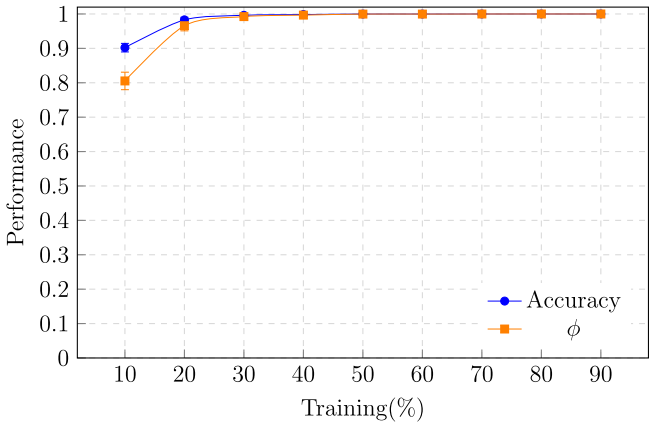


FIG. 12. Training and validating two classes: **Q4** and **Q5**. We generate (102 + 138) matrices. There are 6208 1's and 6154 0's. The method is NB.

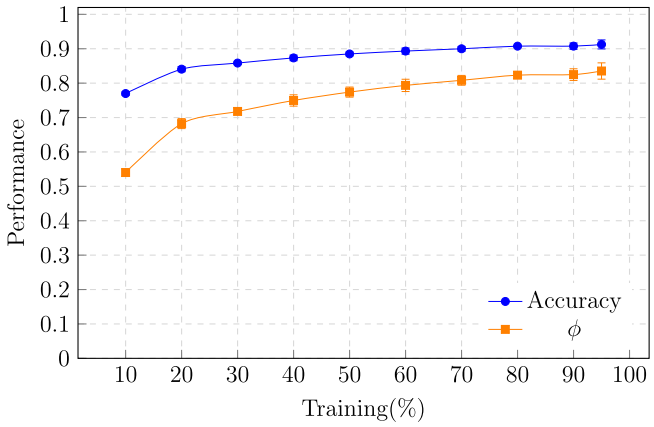


FIG. 13. Training and validating three classes: ['A,' 6], ['D,' 6] and ['E,' 6]. We generate (76 + 77 + 77) matrices. There are 6122 1's and 6090 0's. The method is NB.

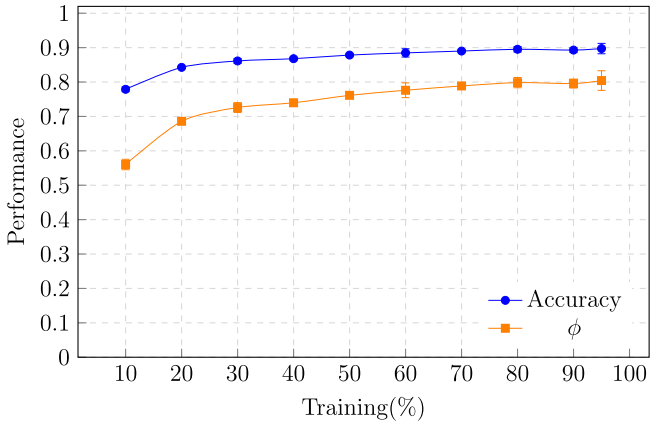


FIG. 14. Training and validating three classes: **Q4**, **Q5**, and **Q6**. We generate (102 + 138 + 161) matrices. There are 11966 1's and 11494 0's. The method is NB.

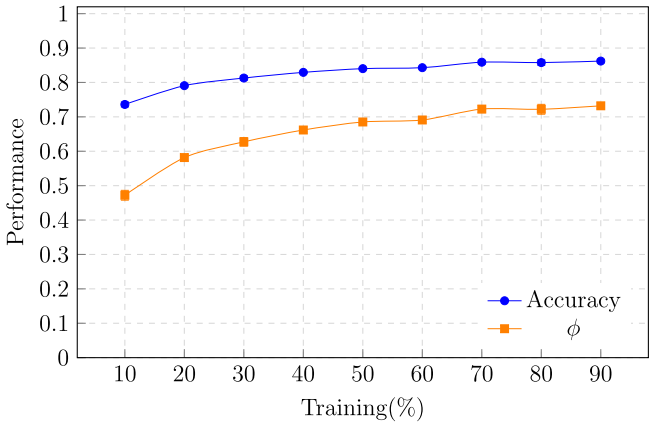


FIG. 15. Training and validating four classes: **Q4**, **Q5**, **Q6**, and **Q7**. We generate $(102 + 138 + 161 + 102)$ matrices. There are 16059 1's and 16250 0's. The method is NB.

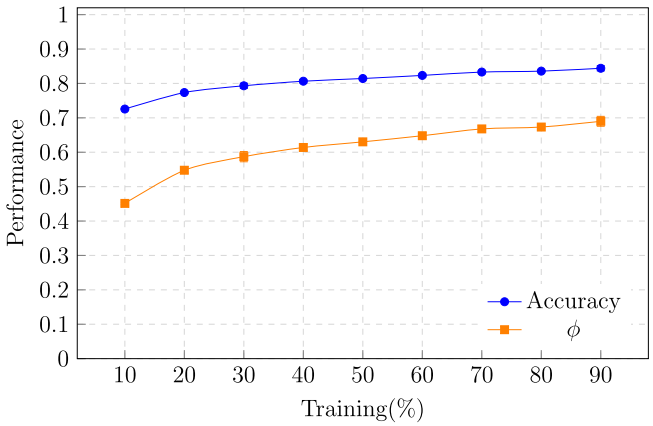


FIG. 16. Training and validating five classes: **Q4**, **Q5**, **Q6**, **Q7**, and **Q8**. We generate $(102 + 138 + 161 + 102 + 161)$ matrices. There are 23377 1's and 23442 0's. The method is NB.

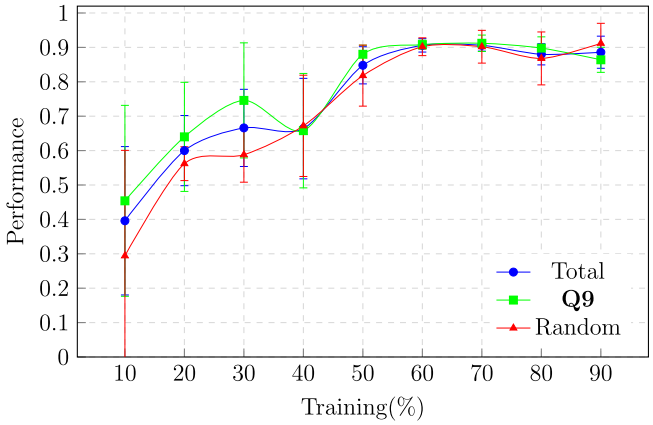


FIG. 17. Training and validating one class, **Q9**, with random matrices. We have $(382 + 388)$ matrices. We use a NN classifier. The learning curves are all accuracies.

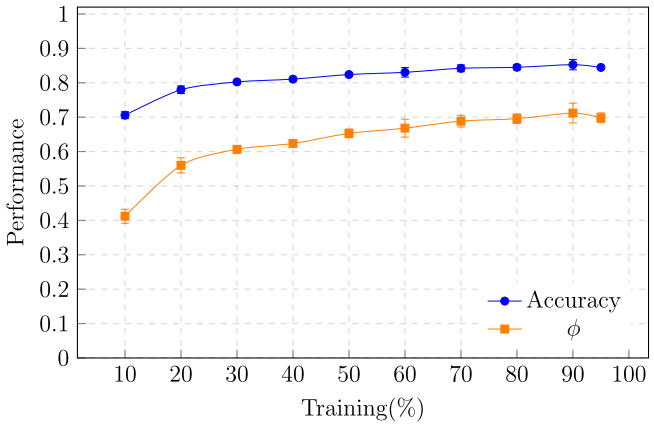


FIG. 18. Training and validating four classes: ['A,' 4], ['D,' 4], ['A,' (3,1),1], and ['A,' (2,2),1]. We generate $(52 + 50 + 70 + 54)$ matrices. There are 5503 1's and 5512 0's. The method is NB.

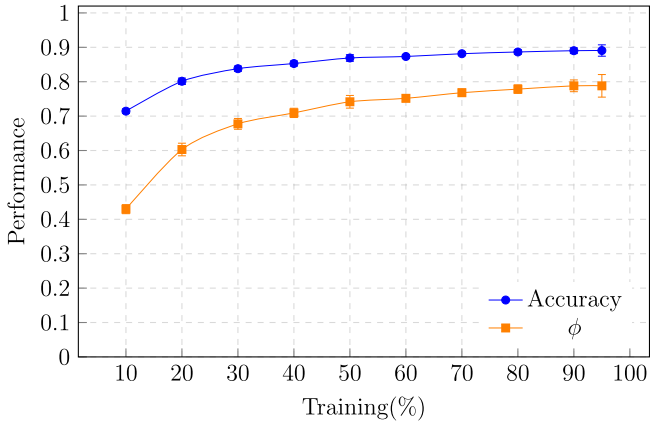


FIG. 19. Training and validating five classes: ['A,' 4], ['D,' 4], ['A,' 6], ['D,' 6] and ['E,' 6]. We generate $(52 + 50 + 76 + 77 + 77)$ matrices. There are 6699 1's and 6711 0's. The method is NB.

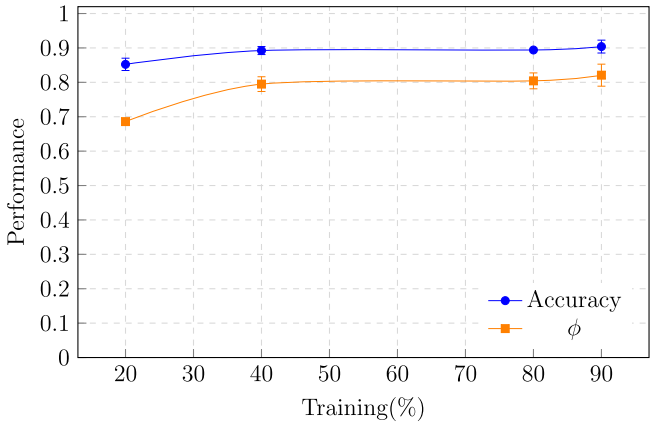


FIG. 20. Training and validating three classes: ['T,' (4,4,4)], ['T,' (4,5,3)], and ['T,' (4,6,2)]. We generate $(65 + 65 + 66)$ matrices. There are 2476 1's and 2301 0's. The method is NB.

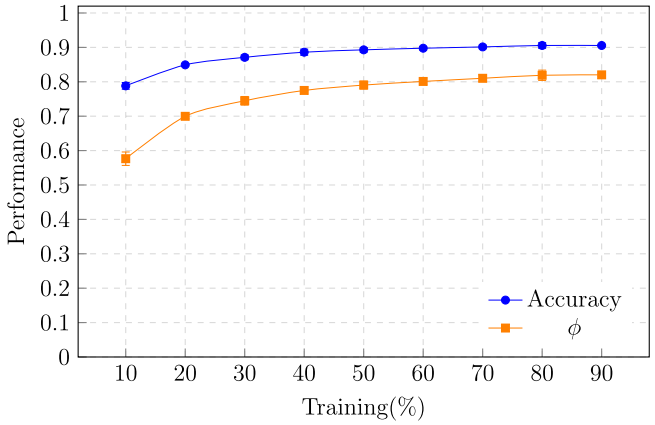


FIG. 21. Training and validating three classes: **Q4**, **Q5**, and **Q6**. We generate $(102 + 138 + 161)$ matrices. There are 11506 1's and 11645 0's. The method is NB. The rank information is included.

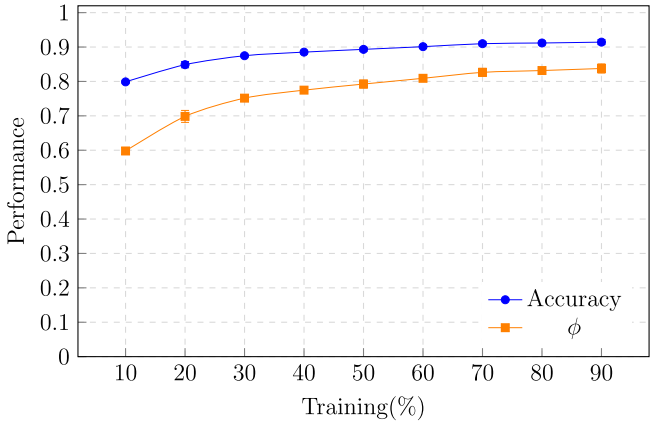


FIG. 24. Training and validating three classes: **Q4**, **Q5**, and **Q6**. We generate $(102 + 138 + 161)$ matrices. There are 11490 1's and 11449 0's. The method is NB. The Diophantine variables are included.

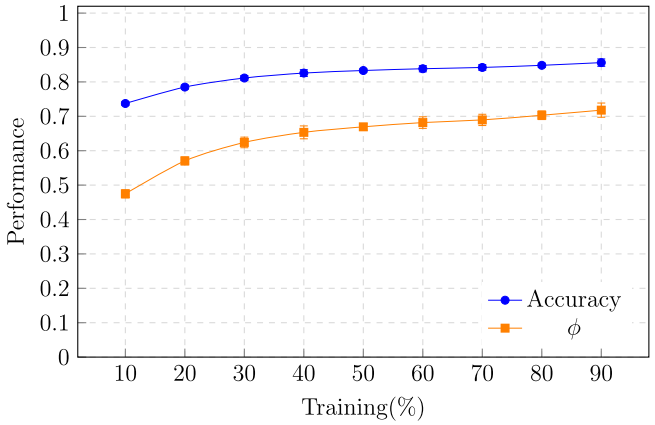


FIG. 22. Training and validating four classes: **Q4**, **Q5**, **Q6**, and **Q7**. We generate $(102 + 138 + 161 + 102)$ matrices. There are 13930 1's and 14005 0's. The method is NB. The rank information is included.

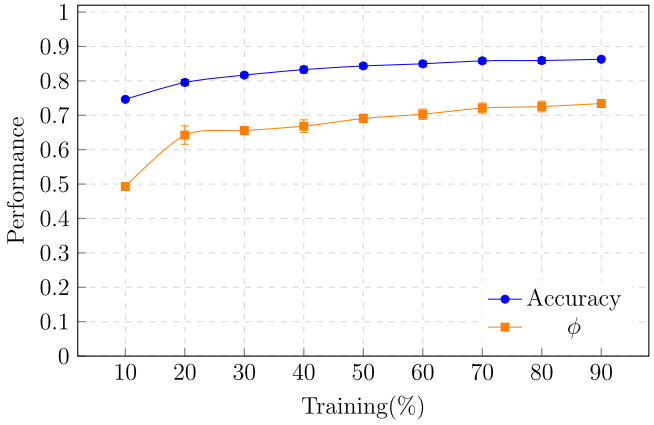


FIG. 25. Training and validating four classes: **Q4**, **Q5**, **Q6**, and **Q7**. We generate $(102 + 138 + 161 + 102)$ matrices. There are 14099 1's and 14118 0's. The method is NB. The Diophantine variables are included.

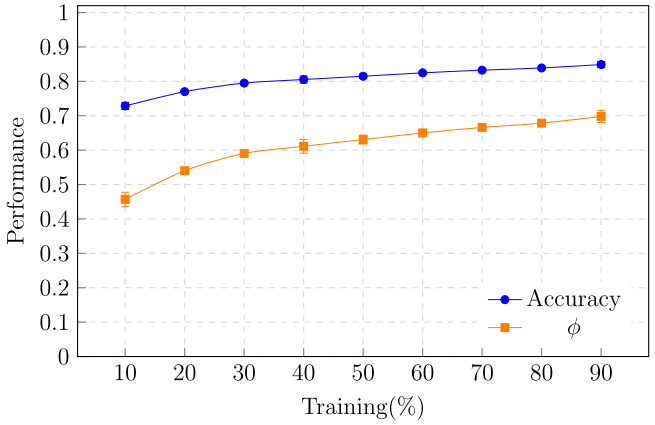


FIG. 23. Training and validating five classes: **Q4**, **Q5**, **Q6**, **Q7**, and **Q8**. We generate $(102 + 138 + 161 + 102 + 161)$ matrices. There are 22770 1's and 22823 0's. The method is NB. The rank information is included via imposing the null vectors.

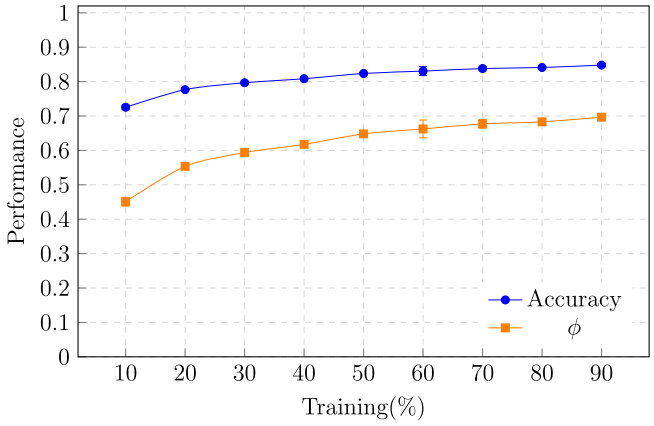


FIG. 26. Training and validating five classes: **Q4**, **Q5**, **Q6**, **Q7**, and **Q8**. We generate $(102 + 138 + 161 + 102 + 161)$ matrices. There are 23273 1's and 24217 0's. The method is NB. The Diophantine variables are included.

See Figs. 12–26. Each graph shows the performance of the investigation's classification method on the specified dataset for varying proportional splits of the dataset into training and validation data. Measures of classification performance considered were accurate, as well as Matthew's correlation coefficient, ϕ, as discussed in Sec. A 4.

[1] N. Seiberg, Electric-magnetic duality in supersymmetric non-Abelian gauge theories, Nucl. Phys. **B435**, 129 (1995).

[2] S. Fomin and A. Zelevinsky, Cluster algebras. I. Foundations, J. Am. Math. Soc. **15**, 497 (2002).

[3] S. Fomin and A. Zelevinsky, Cluster algebras. II. Finite type classification, Inventiones Mathematicae **154**, 63 (2003).

[4] B. Feng, A. Hanany, and Y.-H. He, D-brane gauge theories from toric singularities and toric duality, Nucl. Phys. **B595**, 165 (2001).

[5] B. Feng, A. Hanany, Y.-H. He, and A. M. Uranga, Toric duality as Seiberg duality and brane diamonds, J. High Energy Phys. 12 (2001) 035.

[6] F. Cachazo, B. Fiol, K. A. Intriligator, S. Katz, and C. Vafa, A geometric unification of dualities, Nucl. Phys. **B628**, 3 (2002).

[7] G. Beaujard, J. Manschot, and B. Pioline, Vafa-Witten invariants from exceptional collections, arXiv:2004.14466.

[8] A. Hanany and K. D. Kennaway, Dimer models and toric diagrams, arXiv:hep-th/0503149.

[9] S. Franco, A. Hanany, K. D. Kennaway, D. Vegh, and B. Wecht, Brane dimers and quiver gauge theories, J. High Energy Phys. 01 (2006) 096.

[10] S. Franco, A. Hanany, D. Martelli, J. Sparks, D. Vegh, and B. Wecht, Gauge theories from toric geometry and brane tilings, J. High Energy Phys. 01 (2006) 128.

[11] B. Feng, Y.-H. He, K. D. Kennaway, and C. Vafa, Dimer models from mirror symmetry and quivering amoebae, Adv. Theor. Math. Phys. **12**, 489 (2008).

[12] S. Benvenuti, B. Feng, A. Hanany, and Y.-H. He, Counting BPS operators in gauge theories: Quivers, syzygies and plethystics, J. High Energy Phys. 11 (2007) 050.

[13] S. Fomin, L. Williams, and A. Zelevinsky, Introduction to Cluster Algebras. Chapters 1-3, arXiv:1608.05735.

[14] J. L. Bourjaily, S. Franco, D. Galloni, and C. Wen, Stratifying on-shell cluster varieties: The geometry of non-planar on-shell diagrams, J. High Energy Phys. 10 (2016) 003.

[15] N. Arkani-Hamed, S. He, T. Lam, and H. Thomas, Binary geometries, generalized particles and strings, and cluster algebras, arXiv:1912.11764.

[16] V. V. Fock and A. B. Goncharov, Cluster ensembles, quantization and the dilogarithm, Ann. Sci. École Norm. Supér. Ser. 4 **42**, 865 (2009).

[17] S. Franco and G. Musiker, Higher cluster categories and QFT dualities, Phys. Rev. D **98**, 046021 (2018).

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

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

[20] Y.-H. He, The Calabi-Yau landscape: From geometry, to physics, to machine-learning, arXiv:1812.02893.

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

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

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

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

[25] P. Betzler and S. Krippendorf, Connecting dualities and machine learning, Fortschr. Phys. **68**, 2000022 (2020).

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

[27] R. Altman, J. Carifio, J. Halverson, and B. D. Nelson, Estimating Calabi-Yau hypersurface and triangulation counts with equation learners, J. High Energy Phys. 03 (2019) 186.

[28] M. Demirtas, C. Long, L. McAllister, and M. Stillman, The Kreuzer-Skarke axiverse, J. High Energy Phys. 04 (2020) 138.

[29] Y.-H. He, V. Jejjala, and L. Pontiggia, Patterns in Calabi–Yau distributions, Commun. Math. Phys. **354**, 477 (2017).

[30] T. W. Grimm, F. Ruehle, and D. van de Heisteeg, Classifying Calabi-Yau threefolds using infinite distance limits, arXiv:1910.02963.

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

[32] K. Hashimoto, S. Sugishita, A. Tanaka, and A. Tomiya, Deep learning and the AdS/CFT correspondence, Phys. Rev. D **98**, 046019 (2018).

[33] L. B. Anderson, X. Gao, J. Gray, and S.-J. Lee, Fibrations in CICY threefolds, J. High Energy Phys. 10 (2017) 077.

[34] Y.-H. He and S.-J. Lee, Distinguishing elliptic fibrations with AI, Phys. Lett. B **798**, 134889 (2019).

[35] C. R. Brodie, A. Constantin, R. Deen, and A. Lukas, Machine learning line bundle cohomology, Fortschr. Phys. **68**, 1900087 (2020).

[36] V. Jejjala, A. Kar, and O. Parrikar, Deep learning the hyperbolic volume of a knot, Phys. Lett. B **799**, 135033 (2019).

[37] A. Mütter, E. Parr, and P. K. Vaudrevange, Deep learning in the heterotic orbifold landscape, Nucl. Phys. **B940**, 113 (2019).

[38] R. Deen, Y.-H. He, S.-J. Lee, and A. Lukas, Machine learning string standard models, CERN Reports No. CERN-TH-2020-050 and No. CTPU-PTC-20-06, 2020.

[39] Y. Gal, V. Jejjala, D. K. M. Pena, and C. Mishra, Baryons from mesons: A machine learning perspective, arXiv:2003.10445.

[40] A. Ashmore, Y.-H. He, and B. A. Ovrut, Machine learning Calabi-Yau metrics, arXiv:1910.08605.
[41] Y.-H. He and M. Kim, Learning algebraic structures: Preliminary investigations, arXiv:1905.02263.
[42] Y.-H. He, E. Hirst, and T. Peterken, Machine-learning Dessins d'Enfants: Explorations via modular and Seiberg-Witten curves, arXiv:2004.05218.
[43] L. Alessandretti, A. Baronchelli, and Y.-H. He, Machine learning meets number theory: The data science of Birch-Swinnerton-Dyer, arXiv:1911.02008.
[44] The Sage Developers, SageMath, The Sage mathematics software system (Version 9.0), 2019.
[45] G. Musiker and C. Stump, A compendium on the cluster algebra and quiver package in sage, arXiv:1102.4844.
[46] D. Kutasov and A. Schwimmer, On duality in supersymmetric Yang-Mills theory, Phys. Lett. B **354**, 315 (1995).
[47] D. Kutasov, A comment on duality in $N = 1$ supersymmetric nonAbelian gauge theories, Phys. Lett. B **351**, 230 (1995).
[48] A. Kapustin, The Coulomb branch of $N = 1$ supersymmetric gauge theory with adjoint and fundamental matter, Phys. Lett. B **398**, 104 (1997).
[49] A. Berenstein, S. Fomin, and A. Zelevinsky, Cluster algebras. III. Upper bounds and double Bruhat cells, Duke Math. J. **126**, 1 (2005).
[50] P. Sherman and A. Zelevinsky, Positivity and canonical bases in rank 2 cluster algebras of finite and affine types, Moscow Math. J. **4**, 947 (2004).
[51] P. Gabriel, Unzerlegbare darstellungen. I, Manuscr. Math. **6**, 71 (1972); Erratum, Manuscr. Math. **6**, 309 (1972).
[52] A. Felikson, M. Shapiro, and P. Tumarkin, Skew-symmetric cluster algebras of finite mutation type, J. Eur. Math. Soc. **14**, 1135 (2012).
[53] H. Derksen and T. Owen, New graphs of finite mutation type, Electron. J. Comb. **15**, R139 (2008).
[54] M. Alim, S. Cecotti, C. Córdova, S. Espahbodi, A. Rastogi, and C. Vafa, BPS quivers and spectra of complete $N = 2$ quantum field theories, Commun. Math. Phys. **323**, 1185 (2013).
[55] S. Fomin, M., and D. Thurston, Cluster algebras and triangulated surfaces. Part I: Cluster complexes, arXiv:math/0608367.
[56] V. G. Kac, Infinite root systems, representations of graphs and invariant theory, Inventiones Mathematicae **56**, 57 (1980).
[57] B. Feng, A. Hanany, Y.-H. He, and A. Iqbal, Quiver theories, soliton spectra and Picard-Lefschetz transformations, J. High Energy Phys. 02 (2003) 056.
[58] S. Franco, A. Hanany, Y.-H. He, and P. Kazakopoulos, Duality walls, duality trees and fractional branes, arXiv:hep-th/0306092.
[59] M. Abadi *et al.*, TensorFlow : Large-scale machine learning on heterogeneous systems, 2015.
[60] S. Benvenuti and A. Hanany, New results on superconformal quivers, J. High Energy Phys. 04 (2006) 032.
[61] A. Hanany, Y.-H. He, C. Sun, and S. Sypsas, Superconformal block quivers, duality trees and diophantine equations, J. High Energy Phys. 11 (2013) 017.
[62] S. Franco and A. Hanany, Toric duality, Seiberg duality and Picard-Lefschetz transformations, Fortschr. Phys. **51**, 738 (2003).
[63] S. Franco, A. Hasan, and X. Yu, On the classification of duality webs for graded quivers, J. High Energy Phys. 06 (2020) 130.
[64] A. Gadde, S. Gukov, and P. Putrov, (0, 2) trialities, J. High Energy Phys. 03 (2014) 076.
[65] S. Franco, S. Lee, and R.-K. Seong, Brane brick models and 2d (0, 2) triality, J. High Energy Phys. 05 (2016) 020.
[66] S. Franco, S. Lee, R.-K. Seong, and C. Vafa, Quadrality for supersymmetric matrix models, J. High Energy Phys. 07 (2017) 053.
[67] J. H. Smith, *Some properties of the spectrum of a graph, in Combinatorial Structures and their Applications (Proc. Calgary Internat. Conf., Calgary, Alta., 1969)* (Gordon and Breach, New York, 1970), pp. 403–406.
[68] W. R. Inc., Mathematica, Version 12.0.
[69] H. Zhang, The optimality of naive bayes, in *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference, FLAIRS* (The AAAI Press, Menlo Park, 2004), Vol. 2, p. 01.
[70] K. O'Shea and R. Nash, An introduction to convolutional neural networks, arXiv:1511.08458.
[71] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv:1412.6980.
[72] R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection, in *IJCAI95: Proceedings of the 14th International Joint Conference on Artificial Intelligence* (Morgan Kaufmann Publishers Inc.,, San Francisco, 1995), Vol. 2, pp. 1137–1143.
[73] D. Chicco and G. Jurman, The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation, BMC Genomics **21**, 6 (2020).