Imperial College London
Department of Mathematics

Fano Varieties and Machine Learning

Sara Veneziale

Supervised by
Professor Tom Coates
Dr Alexander M. Kasprzyk

Thesis submitted for the degree of
Doctor of Philosophy in Mathematics
of the Imperial College London,
July 1, 2024

Statement of Originality

I certify that this thesis, and the research to which it refers, are the product of my own work, and that any ideas or quotations from the work of other people, published or otherwise, are fully acknowledged in accordance with the standard referencing practices of the discipline. In particular Chapters 3 and 5 outline joint work with Tom Coates and Alexander M. Kasprzyk [CKV23b] and [CKV24] respectively. Some of the material in Chapter 2 Section 2.5 overlaps with the content of the author's survey article [Ven24a].

Copyright Statement

In memoriam Avi

Abstract

Algebraic geometry is the study of geometrical shapes defined as solutions to polynomial equations – *algebraic varieties*. Amongst varieties, the positively curved ones – *Fano varieties* – distinguish themselves for their importance: they can be considered the building blocks of algebraic geometry. A new approach to the classification of Fano varieties comes from mirror symmetry and involves analysing an invariant called the *regularized quantum period*. This is a sequence of integers which gives a numerical fingerprint for a Fano variety – it is conjecture to determine a Fano variety uniquely. In this thesis, we approach questions related to the classification of Fano varieties in this flavour using both traditional mathematics and a novel methodology coming from data analysis and machine learning. The two main contributions of this thesis are as follows.

In the first part of this thesis, we build accurate machine learning models that predict the dimension of certain Fano varieties from their regularized quantum period. Guided by the model, we are able to establish rigorous asymptotic formulae for the regularized quantum period that make the dependence on the dimension clear. This result is proof-of-concept of two main ideas: firstly how the machine learning models can guide intuition in formulating and proving rigorous mathematical statement. Moreover, it gives positive evidence for the conjecture that the regularized quantum period carries geometric information about the variety.

In the second part of this thesis, we use machine learning differently. When classifying Fano varieties, we note that the objects that we are after are not smooth and well-behaved but have to admit some bad points, called *terminal singularities*. Being able to verify such a condition is of fundamental importance, but can be very challenging. We develop an accurate neural network classifier that predicts whether certain Fano varieties (belonging to a family of geometries that are nicely behaved) have terminal singularities. We use this classifier to accelerate computations by substituting expensive computer algebra routines. This allows us to generate a big dataset to give the first sketch of the landscape of this class of Fano varieties. Moreover, inspired by the machine learning analysis, we formulate and prove a new global combinatorial criterion for checking if this class of Fano varieties has terminal singularities. Together with the first sketch of the landscape of this family of Fano varieties, this gives strong new evidence that machine learning can be an essential tool in developing mathematical conjectures and accelerating theoretical discovery.

Acknowledgements

Firstly, I would like to thank Tom Coates and Al Kasprzyk. They have been more than just supervisors, but teachers, mentors, and collaborators. I thank them for their generosity in sharing their ideas and their enthusiasm about the results. It has been an incredible pleasure to learn from and alongside them these past four years.

I thank my examiners Neil Lawrence and Paolo Cascini, for reading this thesis carefully and for the interesting discussions that originated from it. I would also like to thank Alessio Corti, Anthea Monod, Challenger Mishra, Hamid Abban, John Aston for useful conversations and feedback in different occasions.

I thank the Fanosearch research group, Hannah, Robert, and Thamarai, for creating a nice environment to share ideas in. I would also like to thank the Thesis Family, Arne, Inés, Paul, Qi, Roan, Yueqi and many others, for welcoming me in the past year.

I would like to thank the geometry (and sometimes not geometry) group at Imperial College and across London, for the many lunches and for occasional feedback about talks, papers, and scribbles. Specifically, I would like to thank Ilaria, Marta, Martin, Michela, Nick, Noah, Pascale, Samuele, Stefania. I thank all the cohorts of the LSGNT for putting up with my teaching through various iterations of the Computing Course.

I thank my family who have always been an example of dedication and good work, and all my friends across different countries. I also would like to thank my London family of flatmates, Alexia, Anna, and Ashok, for listening to all the PhD struggles over the years.

And lastly, I thank James for the unwavering emotional (and sometimes mathematical) support.

I acknowledge the support of the Engineering and Physical Sciences Research Council [EP/S021590/1], the EPSRC Centre for Doctoral Training in Geometry and Number Theory (The London School of Geometry and Number Theory), University College London, Imperial College London, and King's College London.

Contents

Introduction

Algebraic geometry studies *algebraic varieties*, high-dimensional geometric shapes that are defined as solutions to polynomial equations. These objects arise naturally and have applications across mathematics and science (for example in coding theory [vLvdG88], cryptography [NX09], string theory [AHDM78], optimisation problems [FRPM06], and algebraic statistics [ERSS05]). A central objective in algebraic geometry is classifying algebraic varieties. The meaning of classifying can change depending on the flavour of geometry we are looking at, but in our context we consider classifying algebraic varieties in the sense of the Minimal Model Program (MMP) [KM98, Kol87]. The MMP describes the process of decomposing a variety into basic pieces, which can be of three types: positively curved (*Fano*), flat (*Calabi–Yau*), negatively curved (*general type*).

Let us consider the example of algebraic curves, also called *Riemann surfaces*; see Table 1. These are one-dimensional geometric shapes over the complex numbers, and they are determined completely by their genus. The projective line (the sphere) has genus zero, and it admits a metric with positive curvature; this is the only one-dimensional Fano variety. All elliptic curves (tori) belong to the same family: they have genus one and they admit a metric with zero curvature. These are the flat examples, or Calabi–Yau. And lastly, we have different families of hyperelliptic curves, which have genus greater than one and admit a metric with negative curvature. These are the general type examples. The MMP can be thought of as a generalisation of this classification of algebraic varieties to dimension greater than one.

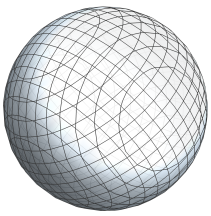
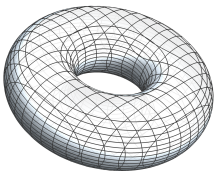
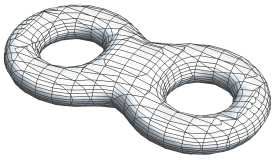
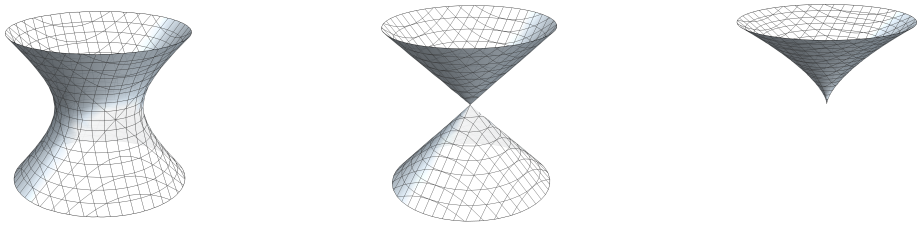
	Sphere/ $\mathbb{P}^1$	**Tori/ Elliptic Curves**	**Hyperelliptic Curves**
			
Genus	0	1	> 1
Curvature	Positive	Zero	Negative

Table 1: Classification of algebraic curves according to genus.

The correct notion of equivalence for classifying varieties in this setting is *deformation*. Intuitively, two algebraic varieties are deformation equivalent if we can change the equation of one to obtain the other by perturbing the coefficients. For example, we can write all elliptic curves as solution to an equation of the form $y^2 = x^3 + ax + b$ for some coefficients a and b. As we change the coefficients, we change some aspects of the geometry of the variety, but we keep the topology the same, meaning that they belong to the same family. When looking at classification questions in algebraic geometry, we will consider deformation families of algebraic varieties rather than the single varieties themselves.

Fano varieties

In the context of the MMP, Fano varieties are – in a precise sense – the *atomic pieces* of algebraic geometry. Not only they are important because they are the positively-curved pieces, but the other two classes (Calabi–Yau and general type) can be realised in a Fano variety by imposing extra equations. Therefore, the classification of Fano varieties (up to deformation) can be thought of as building a *Periodic Table* for geometry. However, despite its fundamental importance, their classification has been open since the 1930s. It is known only for the low-dimensional smooth cases. The only one-dimensional smooth example is the projective line – we have seen this in the classification of algebraic curves. In dimension two, there are ten more deformation families, the Del Pezzo surfaces [DP87]. In dimension three, we jump to 105 families, whose classification combines work of Fano in the 1930s, Iskovskikh in the 1970s, and Mori–Mukai in the 1980s [Fan47, Isk77, Isk78, Isk79, MM82, MM03]. Unfortunately, there is no complete classification for dimensions higher than four, despite knowing that there are finitely many smooth deformation families in each dimension [KMM92]. Even less is known when the varieties are not required to be smooth, despite the fact that there are finitely many families if the singularities are of bounded complexity [Bir21]. In the context of decomposing varieties into their basic pieces, smoothness is actually not the right restriction to impose (examples of smooth and singular algebraic varieties are shown in Figure 1). In fact the building blocks which arise from the MMP are not necessarily smooth, but satisfy a weaker condition, called $\mathbb{Q}$-*factoriality*, and admit mild singularities, called *terminal singularities* [Cas21, Rei87]. We use the term $\mathbb{Q}$-Fano to refer to Fano varieties that satisfy these conditions. These types of singularities are the most restrictive class that needs to be allowed to for the MMP to run.



(a) $x^2+y^2 = z^2+1$ (b) $x^2+y^2 = z^2$ (c) $x^2+y^2 = z^3$

Figure 1: Algebraic varieties in $\mathbb{R}^3$ with different defining equations: (a) is smooth, (b) and (c) havesingularity at the origin.

Besides being the building blocks of algebraic geometry, $\mathbb{Q}$-Fano varieties are interesting in other contexts as well. They play an important role in the study of K-stability and the existence of Kähler–Einstein metrics [Ber16]. On the other hand, they also find applications in theoretical physics. In fact, the main construction of Calabi–Yau manifolds, which gives geometric models of spacetime in Type II string theory [Pol05, Gre97, CHSW85], are through 'anticanonical sections' of $\mathbb{Q}$-Fano varieties. Finally, terminal singularities (which we will focus on the second part of this thesis) not only appear in the Minimal Model Program [Kol87], but also in other areas of mathematics. For example, they find applications in F-theory, where they represent the presence of localized matter states for those M2-branes which are not charged under any massless gauge potential [AGW18]. In addition to this, in the toric case, a variety having at worst terminal singularities is equivalent to considering lattice polytopes that only have one lattice point in their interior, so-called *one-point lattice polytopes*. These objects are important in optimisation problems.

Mirror symmetry

A new approach to the classification of Fano varieties has arisen using ideas coming from *mirror symmetry*. Mirror symmetry is a new and active area of mathematics which has its roots in string theory. This term was originally used to refer to the correspondence that has been observed between certain Calabi–Yau varieties by theoretical physicists and algebraic geometers; see [CHSW85, SYZ96]. Since then, the term mirror symmetry has been used in many contexts to refer to a wide system of far-reaching conjectures of a duality between different geometric objects. In our case, mirror symmetry for Fano varieties refers to the conjectured correspondence between (deformation classes of) Fano varieties and (mutation classes) of certain Laurent polynomials [CCG$^+$13]. Both these objects can be associated to certain power series (the *regularized quantum period* for Fano varieties, and the *classical period* for Laurent polynomials). If these two power series coincide, the two objects are said to be mirror partners.

Given a Fano variety X, its regularized quantum period is given as

$$\hat{G}_X(t) = \sum_{d=0}^{\infty} c_d t^d \,,$$

where $c_0 = 1$, $c_1 = 0$, $c_d = r_d d!$, where r_d are certain Gromov–Witten invariants of X. Intuitively, each r_d counts the number of rational curves of degree d in X that pass through a fixed generic point and have a certain constraint on their complex structure. We call *period sequence* the sequence $(c_d)_d$ of coefficients of the regularized quantum period. This object not only arises in the study of mirror symmetry for Fano varieties, but it is expected to contain a lot of geometric information about the variety. In fact, it is conjectured to be a *complete invariant* of Fano varieties up to deformation. This is proven in the smooth low-dimensional case, where we have a complete classification, and there are no known counterexamples [CCGK16]. Regardless of whether the conjecture is true, the regularized quantum period remains an extremely fine invariant for Fano varieties. One of the major motivating question behind this thesis is to explore how to extract geometric information about a variety from its regularized quantum period.

A machine learning approach

The central idea of this thesis is to use data analysis and machine learning, in combination with geometric and combinatorial tools, to explore the landscape of Fano varieties, with the ultimate aim to formulate and prove rigorous statements. The content of this work positions itself in the new wave of application of machine learning to pure mathematics [DVB+21, He23, EF21, WDL22, Wag21] and answers affirmatively to the question 'can machine learning bring insight into pure mathematics research?'. We note that whilst this is a novel approach, it fits well in the paradigm of how to do mathematics. Pattern recognition has been at the core of mathematical discovery for centuries. The most famous examples are perhaps the conjecture of the Prime Number Theorem by Gauss and the formulation of the Birch and Swinnerton-Dyer Conjecture in the 1960s [BSD63]. The latter is also a prominent example of the use of computer experiments to drive conjecture formulation. More generally, computer assisted experiments in this flavour have led to a growth in interest and availability of big datasets of mathematical objects; see for example the Atlas of Lie Groups [APC+16], Kreuzer and Skarke's dataset of reflexive polyhedra [KS00], the Atlas of finite groups [CCN+85], and Cremona's dataset of L-functions [Cre16]. Overall this increase in development and use of computational tools in pure mathematics is now being complemented by a new emerging methodology that studies mathematical objects using tools from data analysis and machine learning. In particular, in this work we present two paradigmatic workflows of interplay between machine learning techniques and pure mathematics. The first one showcases how high-accuracy machine learning models can be used to guide mathematician's intuition in conjecturing (and then proving) mathematical statements. On the other hand, we also give an example of how indispensable machine learning can be in accelerating mathematical discovery by taking the place of expensive computer algebra routines in data generation pipelines.

Contribution

In this thesis we summarise the contents of [CKV23b, CKV24] that showcase the interplay between pure mathematics and the machine learning methodology. Our problem is suitable for a machine learning framework for numerous reasons. Firstly, the sheer amount of data makes this problem hard to navigate on paper, but having many datapoints is the area in which machine learning methods really shine. In addition to this, we will probe the landscape of $\mathbb{Q}$-Fano varieties by analysing highly symmetrical objects – toric varieties – whose geometry is controlled by combinatorics and therefore are suitable examples to handle in a computational context. The chapters dealing with these themes are complemented by purely theoretical upcoming work, whose construction has relied heavily on computational tools.

Toric Fano varieties Let us explicitly describe the objects we are considering – toric Fano varieties. The prototypical example of this class of objects is *projective space* $\mathbb{P}^n$ which can be defined as the quotient $\mathbb{C}^{n+1} \setminus \{\mathbf{0}\}/\mathbb{C}^\times$ under the action

$$\lambda \cdot (z_0, \ldots, z_n) = (z_0, \ldots, z_n) \,.$$

This can be generalised to actions that have different weights, such as

$$\lambda \cdot (z_0, \ldots, z_n) = (\lambda^{a_0} z_0, \ldots, \lambda^{a_n} z_n)$$

whose quotient gives rise to a *weighted projective space* $\mathbb{P}(a_0, \ldots, a_n)$.

For a general toric Fano of Picard rank r and dimension $N - r$, its geometry is described by an integer–valued matrix describing an action of $(\mathbb{C}^\times)^r$ on $\mathbb{C}^N$, whose geometric quotient is the variety. In our context, we will mostly concentrate on toric Fano varieties of Picard rank two, whose data is summarised in a $2 \times N$ integer–valued matrix

$$\begin{bmatrix} a_1 & a_2 & \cdots & a_N \\ b_1 & b_2 & \cdots & b_N \end{bmatrix} \tag{0.0.1}$$

which describes the action $(\mathbb{C}^\times)^2$ on $\mathbb{C}^N$

$$(\lambda, \mu) \cdot (z_1, \ldots, z_N) = (\lambda^{a_1} \mu^{b_1} z_1, \ldots, \lambda^{a_N} \mu^{b_N} z_N)\,.$$

Toric Fano varieties have many other benefits which makes them amenable to the machine learning tools we use in this thesis. For example, they have a rich combinatorial structure which captures their geometric properties. Therefore, they work well within computer algebra systems and automated data generation pipelines. Moreover, in the context of mirror symmetry we have explicit formulae to compute their regularized quantum periods.

Machine learning for conjecture generation In [CKV23b], we asked if we can extract the dimension of a Fano variety from its period sequence. As previously discussed each coefficient in the sequence *counts* curves in the variety. One can think of the period sequence as a *numerical fingerprint* of a Fano variety, since it is conjectured be an invariant that determines the Fano variety uniquely [CCG+13, CCGK16]. Therefore, it is natural to ask if and how we can extract geometric properties from it. To answer this, in [CKV23b] we built very accurate machine learning classifiers predicting the dimension for two classes of toric Fano varieties (weighted projective spaces and toric varieties of Picard rank two) from their period sequences. Our machine learning pipeline motivated us to discover and prove rigorous asymptotics for these classes' regularized quantum periods, allowing us to extract the dimension from them. Furthermore, we used this to sketch where these classes of toric varieties fall in the landscape of all Fano varieties, highlighting how they cluster by dimension. This gave an important proof-of-concept that the regularized quantum period indeed carries geometric information and showed a lot of mathematical structure waiting to be uncovered and understood.

The asymptotic formulae for these classes are generalised to a wider collection of objects, all toric varieties and toric complete intersections (under some assumptions), in upcoming work [Ven]. We are restricted to these classes since, among Fano varieties, those that are toric or toric complete intersections have nice combinatorial formulae that describe the regularized quantum period [CCGK16]. However, this is not a very strong restriction since, in the known classification of smooth Fano varieties, these represent a big class of examples: all smooth two-dimensional Fano varieties are toric or toric complete intersections, and so are 92 of the 105 in the three-dimensional case [CCGK16]. Moreover, there are many examples of results that were originally proved for toric and toric complete intersections, and

then generalised to less restrictive classes of varieties [Giv96, Giv98, Giv01, GS19, Tel12]. Therefore, behaviour exhibited in these results is hopefully representative of a wider class of Fano varieties.

Machine learning for computations In [CKV24] we built a highly accurate neural network model that determines whether a toric Fano variety is terminal. As highlighted above, being terminal is an essential property, but known terminality tests are often very computationally expensive. For example, just generating the dataset we used in the paper took 30 CPU years. With the machine learning model to guide us, we developed a new algorithm which is 15 times faster than any established method. Moreover, the success of the neural network (which is up to 2000 times faster than the new algorithm) allowed us to quickly generate many examples to explore the landscape of terminal toric Fano varieties: generating ten times the original amount of data took only 120 CPU hours. While exploring the landscape of terminal toric Fano varieties, we discovered a striking dependence between the asymptotics of the regularized quantum period and another invariant called the *Fano index*. We expect this approach to bring even more insight on the Fano classification problem in the future.

In the last part of the thesis, we discuss algorithmic approaches to the study of toric Fano varieties. We start by presenting the new algorithm that checks if a toric Fano variety of Picard rank two has at worst terminal singularities from [CKV24]. We further generalise this to check for canonical singularities, and discuss why this methodology is not straightforward to apply to the cases of Picard rank higher than two. Lastly, we discuss ongoing work building algorithms to test whether a sequence of number can arise as the period sequence of a toric Fano variety. We successfully implement this for weighted projective spaces and a special case of Picard rank two toric Fano varieties. We further discuss generalisations. Building such algorithms is of importance since, in the context of mirror symmetry we can look at Fano varieties through their mirror partners, Laurent polynomials. Recall, a Laurent polynomial f is related to a Fano variety X if its classical period (another sequence) coincides with the regularized quantum period of X. Therefore, being able to determine whether a classical period could arise as the regularized quantum period answers the question of whether a Laurent polynomial is a mirror to a Fano variety. Constructing Laurent polynomials and exploiting this mirror construction has been used to construct new examples of Fano varieties[CKP19, CHK22, Heu22].

Structure of the thesis

Chapter 1 gives the necessary mathematical background on Fano varieties, toric geometry, and mirror symmetry for Fano varieties. In Chapter 2, we introduce the key machine learning algorithms and methodology that will be utilised in the thesis. Moreover, we discuss this novel machine learning approach to mathematical data and survey various applications of machine learning to pure mathematics. Chapter 3 covers the content of [CKV23b] on the application of machine learning techniques to predict the dimension of toric Fano varieties from their period sequences. In Chapter 4 we discuss recent progress in the construction of rigorous asymptotics of the regularized quantum period of toric varieties and toric complete intersections, generalising the result of toric varieties with Picard rank one

and two from the previous chapter. In Chapter 5 we look at the results of [CKV24] where we use machine learning to build a classifier that distinguishes between terminal and non-terminal toric varieties of Picard rank two and dimension eight, and we use this to start exploring the landscape of $\mathbb{Q}$-Fano toric varieties. In Chapter 6 we summarise different results relating to the construction of algorithms for toric Fano varieties. In particular, we discuss a new algorithm to determine whether a Picard rank two toric Fano variety is terminal or not from [CKV24], which was inspired by the machine learning result in Chapter 5. In Chapter 7 we conclude the thesis with some outlook, in which we reflect on our approach and discuss the next steps in integrating pure mathematics and machine learning.

Impact Statement

The impact of this thesis spans a range of scientific areas. Firstly, the immediate impact is in algebraic geometry, specifically to the study of the classification of Fano varieties. This thesis looks at these objects using a machine learning methodology which brings insight into their structure. The classification of Fano varieties is a fundamental problem in geometry, but any further understanding of it is very impactful across other disciplines, especially theoretical physics. In fact, four-dimensional Fano varieties naturally contain three-dimensional Calabi–Yau manifolds, which play a key role in constructing models of spacetime in string theory [Pol05].

Secondly, this thesis impacts pure mathematics more broadly. This work joins [DVB$^+$21] as a notable example of a successful application of machine learning to pure mathematics to drive conjecture generation. In our case this paradigm is applied to geometry, but its impact is potentially much wider. Many mathematical areas have availability of big datasets (see again [BKnt, APC$^+$16, KS00, CK22, CCN$^+$85, Cre16]). Despite this, a machine learning and data analysis focused approach to the study of most of these datasets is still very underexplored. Such experiments can bring insights to these mathematical structures and lead to the conjecture and proof of new mathematical statements.

Thirdly, this thesis has the potential to impact pure machine learning research as well. Mathematical data does not have outliers or noise. Therefore, mathematical datasets can be used as a testbed for machine learning methodologies. For example, the ability (or inability!) of deep learning algorithms to achieve high accuracy on complicated data structures is an interesting problem. This is especially true in the context of mathematical data, where we know that there should be a precise mathematical explanation underlying a certain phenomenon. Moreover, mathematical data provides interesting examples for machine learning experiments. This is especially true in the context of equivariant and invariant machine learning approaches. Mathematical data often comes with symmetries which are usually much richer than, for example, translation and rotation invariance that is considered in image recognition tasks. Understanding the relationship between machine learning algorithms and symmetrical data for widely general classes of examples will improve and develop the robustness of current approaches.

Conventions Vectors $v \in \mathbb{R}^n$ are considered as column vectors. We are always working over the field $\mathbb{C}$.

Data availability All datasets are available on Zenodo [EO13]. The datasets that we refer to in Chapter 3 are available at [CKV22a, CKV22b] under a CC0 license. The dataset that we refer to in Chapter 5 is available at [CKV23a] under a CC0 license.

Code availability The code used to run the experiments described in Chapter 3 is available on BitBucket at [CKV22c] under an MIT licence. The code used to run the experiments described in Chapter 5 is available on BitBucket at [CKV23c] under an MIT licence. The implementation of algorithms described in Chapters 4 and 6 is available on BitBucket at [Ven24b] under an MIT licence.

1 Mathematical Background

In this chapter we introduce the basics of Fano varieties, toric geometry, and mirror symmetry. This is aimed at non-specialists and assumes minimal algebraic geometry knowledge. We limit ourselves to the objects and concepts that will appear in the rest of the thesis. Nonetheless, we include precise definitions in the footnotes and point the reader to the references when appropriate.

1.1 Toric varieties

Our main objects of study are toric Fano varieties. In this section we introduce toric varieties from the point of view of *Geometric Invariant Theory* (GIT). We point the reader to the references for a broader introduction to toric geometry [Ful93, CLS11]. Toric varieties are a nice class of highly symmetrical algebraic varieties with a rich combinatorial structure. Recall from the introduction that algebraic varieties are those geometric shapes defined as solutions to polynomial equations. For now, the reader can consider the requirement of being Fano as assuming that these varieties are, in some sense, *positively curved* – we will give a precise definition later in the chapter. In the context of toric varieties, we will see that the condition of being Fano can also be given a combinatorial charactisation.

The prototypical example of a toric Fano variety is projective space, which is defined as the quotient $\mathbb{P}^N = (\mathbb{C}^{N+1} \setminus \mathbf{0})/\mathbb{C}^\times$, where the action is given explicitly as

$$\lambda \cdot (z_0, \ldots, z_N) = (\lambda z_0, \ldots, \lambda z_N)\,. \tag{1.1.1}$$

We can think of the data corresponding to $\mathbb{P}^N$ to be represented by the vector $(1, \ldots, 1)$. Explicitly, the exponents of λ in (1.1.1).

We can consider $\mathbb{C}^\times$ acting with *weights* different from one, which then gives rise to a class of objects called *weighted projective spaces*. They arise as quotients $(\mathbb{C}^{N+1} \setminus \mathbf{0})/\mathbb{C}^\times$, where now the action is weighted by $(a_0, \ldots, a_N) \in \mathbb{Z}^N_{>0}$, i.e. it is given by

$$\lambda \cdot (z_0, \ldots, z_N) = (\lambda^{a_0} z_0, \ldots, \lambda^{a_N} z_N)\,.$$

Given any two weighted projective spaces $X = \mathbb{P}(a_0, \ldots, a_N)$ and $Y = \mathbb{P}(b_0, \ldots, b_M)$, their

product, $X \times Y$, is not a weighted projective space anymore, but it is a *toric variety*[1]. The product arises as a quotient of $\mathbb{C}^{N+M+2}$ by an action of $(\mathbb{C}^\times)^2$, where the first $\mathbb{C}^\times$ acts on the first $N+1$ co-ordinates of $\mathbb{C}^{N+M+2}$ and the second $\mathbb{C}^\times$ acts on the last $M+1$ co-ordinates. The action is described by a *weight matrix* (instead of a weight vector as for weighted projective spaces),

$$\begin{bmatrix} a_0 & \cdots & a_N & 0 & \cdots & 0 \\ 0 & \cdots & 0 & b_0 & \cdots & b_M \end{bmatrix} .$$

Explicitly, the action is given by

$$(\lambda, \mu) \cdot (z_1, \ldots, z_{N+1}, z_{N+2}, \ldots, z_{M+N+2}) = (\lambda^{a_0} z_1, \ldots, \lambda^{a_N} z_{N+1}, \lambda^{b_0} z_{N+2}, \ldots, \lambda^{b_M} z_{M+N+2}) .$$

This construction can be generalised to any action of $(\mathbb{C}^\times)^2$ on $\mathbb{C}^N$ (for $N > 2$), which can be given as

$$(\lambda, \mu) \cdot (z_1, \ldots, z_N) = (\lambda^{a_1} \mu^{b_1} z_1, \ldots, \lambda^{a_N} \mu^{b_N} z_N)$$

and can be encoded in a weight matrix of the form

$$\begin{bmatrix} a_1 & \cdots & a_N \\ b_1 & \cdots & b_N \end{bmatrix} , \tag{1.1.2}$$

where all $\left(\begin{smallmatrix} a_i \\ b_i \end{smallmatrix}\right) \in \mathbb{Z}^2 \setminus \mathbf{0}$ and they lie in a strictly convex cone[2].

The result of taking such a quotient is not assured to be a *nice* algebraic variety. In fact, the toric varieties we are after are defined as the geometric quotient of $\mathbb{C}^N$ by $(\mathbb{C}^\times)^2$, where *geometric* means that we might have to ignore some point of $\mathbb{C}^N$ for the quotient to be well-behaved. In practice, we need to ignore those points whose orbits are not nice. We have actually already seen this in the case of weighted projective spaces, where the origin of $\mathbb{C}^N$ needs to be excluded when taking the quotient by the action of $\mathbb{C}^\times$, since it is the only point whose orbit contains just the origin itself. When we have an action of $(\mathbb{C}^\times)^2$ we need to remove more points from $\mathbb{C}^N$. Which points we need to remove depend on what the action looks like. In practice, the quotient is of $\mathbb{C}^N \setminus S$ by the action of $(\mathbb{C}^\times)^2$, where S is a union of linear subspaces, S_- and S_+ defined as

$$\begin{aligned} S_+ &:= \{(x_1, \ldots, x_N) \mid x_i = 0 \text{ if } b_i/a_i < b/a\} , \\ S_- &:= \{(x_1, \ldots, x_N) \mid x_i = 0 \text{ if } b_i/a_i > b/a\} , \end{aligned} \tag{1.1.3}$$

for $a = \sum_{i=1}^N a_i$ and $b = \sum_{i=1}^N b_i$; see [BCZ04] for details.

Of course, a similar construction can be done for $\mathbb{C}^N$ quotiented by an action of $(\mathbb{C}^\times)^r$ for $r > 2$. This action will be described an $r \times N$ integer–valued weight matrix (whose columns lie in a convex cone). In this case we do not have as nice of an expression for the points to remove as in (1.1.3), but we can still define them explicitly as follows.

Firstly, note that the construction of toric varieties as GIT depends on the choice of a *stability condition*, which is an element in the column space of the weight matrix. In our cases, since

[1] A toric variety is an irreducible algebraic variety which contains a torus $(\mathbb{C}^\times)^n$ as an open subset such that the action of the torus on itself extends to a morphism of varieties $(\mathbb{C}^\times)^n \times X \to X$.

[2] The requirement of the cone to be convex implies that the variety is projective.

we will require all our toric varieties to be Fano (we will detail the definition of Fano later in this chapter), the stability condition is always chosen to be the sum of the columns. This choice of stability condition affects which points of $\mathbb{C}^N$ need to be removed for the quotient to be geometric. For example choosing the stability condition to be the sum of the columns of the matrix (1.1.2) gives (1.1.3) for the points to be removed. Given a stability condition ω we can define

$$\mathcal{A}_\omega = \{I \subset \{1, \dots, N\} \mid \omega \in \angle_I\},$$

where, if $\alpha_i \in \mathbb{Z}^r$ is the ith column of the weight matrix,

$$\angle_I = \left\{ \sum_{i \in I} \lambda_i \alpha_i \mid \lambda_i \in \mathbb{R}_{>0} \right\}.$$

Then, we can define

$$U_\omega = \bigcup_{I \in \mathcal{A}_\omega} (\mathbb{C}^\times)^I \times \mathbb{C}^{\bar{I}}$$

where $\bar{I} = \{1, \dots, N\} \setminus I$. Here, U_ω can be written as $\mathbb{C}^N$ minus a union of linear subspaces. Then, the toric variety is realised as the quotient $U_\omega/(\mathbb{C}^\times)^r$. Clearly, different choices of stability condition ω give rise to different U_ω and in turn different algebraic varieties, which might not be Fano. However, choosing the stability condition to be the sum of the columns of the weight matrix assures it is always Fano. Different quotients arising by taking different stability conditions for the same action are related to each other by birational transformations[3].

Note that given a toric Fano variety, there are multiple weight matrices that can give rise to it, via this GIT quotient construction we have described above. In fact, we have two group actions on a weight matrix that leave the corresponding quotient unchanged. Given a weight matrix $W \in \mathbb{Z}^{r \times N}$, there is an S_N action that permutes the columns of W, and a $\mathrm{GL}_r(\mathbb{Z})$[4] action that reparametrises the basis of the torus, acting by multiplication on the left. Both of these actions change W but leave the corresponding GIT quotient unchanged.

Remark 1.1.1. For the rest of the thesis we will always choose the stability condition to be the sum of the columns of the weight matrix.

1.1.1 From weight matrices to fans

More traditional surveys of toric varieties usually first introduce the concept of fans, rather than the GIT quotient description we have given above. Therefore, we now describe how to construct a fan structure associated to a weight matrix. Note that this process is not unique, just as the choice of weight matrix does not uniquely identify the toric variety.

Let W be a weight matrix as in the last section. We denote the columns of W by $\alpha_i \in \mathbb{Z}^r$. Let ω be the stability condition we have mentioned above, which we take to be the sum of

[3]The process is referred to as *variation of GIT quotients* [DH98].

[4]Recall that $\mathrm{GL}_r(\mathbb{Z})$ is the group of $r \times r$ integer–valued invertible matrices whose inverse is also integer–valued.

the weight matrix columns. Consider the short exact sequence

$$0 \to \mathbb{Z}^r \to \mathbb{Z}^N \to \mathbb{Z}^{N-r} \to 0\,,$$

where the map $\mathbb{Z}^r \to \mathbb{Z}^N$ is given by W^T and $\beta : \mathbb{Z}^N \to \mathbb{Z}^{N-r}$ is its cokernel map. Let $\rho_i \in \mathbb{Z}^{N-r}$ be the primitive images of the standard basis in $\mathbb{Z}^N$ under β and σ_I be the cone generated by $\{\rho_i \mid i \in I\}$. Then, the corresponding fan Σ is the cone complex in $\mathbb{Z}^{N-r}$

$$\Sigma = \{\sigma_I \mid \bar{I} \in \mathcal{A}_\omega\}\,.$$

We will also consider the concept of the *spanning polytope*. This is the convex polytope P obtained as the convex hull of the primitive generators of the rays (i.e. the one-dimensional cones) of the fan Σ.

Example 1.1.2. Consider the weight matrix

$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \end{bmatrix}\,. \tag{1.1.4}$$

Then, we can construct a corresponding fan Σ in $\mathbb{Z}^2$ with primitive generators of the rays $e_1 = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$, $e_2 = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$, $e_3 = \left(\begin{smallmatrix}-1\\-1\end{smallmatrix}\right)$, $e_4 = \left(\begin{smallmatrix}-1\\-3\end{smallmatrix}\right)$, since

$$\begin{pmatrix}1\\0\end{pmatrix} + \begin{pmatrix}0\\1\end{pmatrix} + \begin{pmatrix}-1\\-1\end{pmatrix} = \begin{pmatrix}0\\0\end{pmatrix} \qquad \begin{pmatrix}1\\0\end{pmatrix} + 3\begin{pmatrix}0\\1\end{pmatrix} + \begin{pmatrix}-1\\-3\end{pmatrix} = \begin{pmatrix}0\\0\end{pmatrix}\,.$$

The appropriate fan and (spanning) polytope are pictured in Figure 1.1.

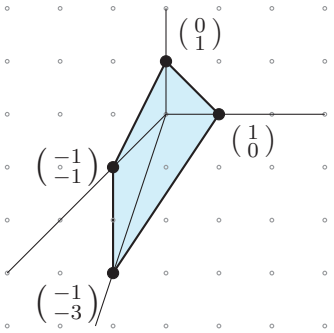


Figure 1.1: Polytope (in blue) and fan corresponding to the toric variety arising from the weight matrix in (1.1.4).

1.1.2 Terminal quotient singularities

As mentioned in the Introduction, in the context of the Minimal Model Program smooth Fano varieties are not the correct *basic pieces* to consider. In fact, the minimal objects need not be smooth. The correct restriction is requiring them to be $\mathbb{Q}$-factorial and admit certain singularities, called *terminal singularities*. This is the most restrictive class of singularities

that has to be allowed in the Minimal Model Program to make it work. We will call these objects $\mathbb{Q}$-Fano varieties.

The definition of $\mathbb{Q}$-factorial for algebraic varieties is more general[5], but for the toric case we have a combinatorial characterisation. Locally, a $\mathbb{Q}$-factorial toric singularity of dimension n is defined as the quotient of affine space $\mathbb{A}^n$ by a finite abelian subgroup of GL_n. For toric Fano varieties, the condition of being $\mathbb{Q}$-factorial is equivalent to checking that the spanning polytope is simplicial.

Terminal singularities[6] were originally introduced by Reid [Rei80]. In dimension two, an algebraic variety has at worst terminal singularities if and only if it is being smooth. In dimension three, we have an analytic classification of terminal singularities [Mor85, Rei87], which specialises to the classification in the toric case. In dimension four we do not even have the toric classification.

In the context of toric Fano varieties, checking whether they have at worst terminal singularities reduces to a combinatorial criterion. For weighted projective spaces we do so using the following results. Note that Proposition 1.1.3 is a necessary and sufficient condition, while Proposition 1.1.4 is only a necessary condition, but it is still useful to rule out many examples.

Proposition 1.1.3 (Proposition 2.3, [Kas13])**.** *Let $X = \mathbb{P}(a_0, \ldots, a_N)$ be a weighted projective space of dimension at least three. Then X has at worst terminal singularities if and only if*

$$\sum_{i=0}^{N} \{k a_i / a\} \in \{2, \ldots, N-2\}$$

for each $k \in \{2, \ldots, a-2\}$. Here $a = a_0 + \cdots + a_N$ and $\{q\}$ denotes the fractional part $q - \lfloor q \rfloor$ of $q \in \mathbb{Q}$.

Proposition 1.1.4 (Theorem 3.5, [Kas09])**.** *Let $X = \mathbb{P}(a_0, \ldots, a_N)$ be a weighted projective space of dimension at least two, with weights ordered $a_0 \leq a_1 \leq \ldots \leq a_N$. If X has at worst terminal singularities then $a_i/a < 1/(N-i+1)$ for each $i \in \{2, \ldots, N\}$.*

A result of this thesis is a generalisation of the criterion from Proposition 1.1.3 to the Picard rank two case, proved in Chapter 6, but otherwise checking if a higher Picard rank variety has at worst terminal singularities uses the fan data rather than the GIT quotient data. Given a D-dimensional toric Fano variety X, let $\Sigma \subset \mathbb{R}^D$ be its fan with primitive rays $e_1, \ldots, e_N$. Then we can write each top-dimensional cone $\sigma \in \Sigma$ as

$$\sigma = \langle e_{i_1}, \ldots, e_{i_D} \rangle_{\mathbb{R}_{\geq 0}} ,$$

i.e. it is generated by a subset of D of the vectors defining the fan (under the assumption of the variety being $\mathbb{Q}$-factorial). Then, the generators of each cone lie in a $(D-1)$-dimensional

[5] An algebraic variety X is *$\mathbb{Q}$-factorial* if it is normal and, in addition, for each rank-one reflexive sheaf E on X, some tensor power of E is a line bundle. This implies that the dimension of the singular locus in X is at most $\dim(X) - 2$, and that some tensor power of the canonical sheaf (of top-degree differential forms) is a line bundle.

[6] A variety X has *terminal singularities* if it satisfies two conditions: there exists $r \in \mathbb{Z}_{\geq 1}$ such that rK_X is Cartier; if $f : X \to Y$ is a resolution of singularities and $\{E_i\}_i$ is the family of all exceptional prime divisors of f then $rK_y = f^*(rK_X) + \sum_i a_i E_i$ for $a_i > 0$.

hyperplane H. Testing terminality for the variety is equivalent to, for each top-dimensional cone in the fan, testing whether the only lattice points in the cone that lie on or below H are the generators of the cone and the origin. In fact, we can say that a toric variety X has at worst terminal singularities if and only if the corresponding spanning polytope P is such that $P \cap \mathbb{Z}^D = \text{vert}(P) \cup \mathbf{0}$.

1.2 Some definitions

In the thesis we will use the concept of Picard rank and Fano index of toric Fano varieties. We recall the general definitions of these, and give a combinatorial characterisation, which is the only thing that will be relevant for the other chapters of this thesis.

1.2.1 The Picard rank

Definition 1.2.1. Given an algebraic variety X, we denote by $\text{Pic}(X)$ its *Picard group*, which is the group of isomorphism classes of line bundles (i.e. one-dimensional vector bundles) on the variety.

Definition 1.2.2. The *Neron–Severi group* is the quotient $\text{Pic}(X)/\text{Pic}^0(X)$, here $\text{Pic}^0(X)$ is the connected component of the identity.

The Neron–Severi group is finitely generated and its rank is called the *Picard rank* of the variety.

In the context of toric Fano varieties given as weight matrices that we are considering in this section, the Picard rank is bounded above by the number of rows of the weight matrix. For example, a weight matrix like (1.1.2) could represent either a Picard rank one variety (a weighted projective space) or a Picard rank two variety. We make precise the conditions needed for our weight matrices to have the Picard rank coincide with the number of rows in Section 1.4. Note that the dimension of a toric variety can be read from its weight matrix representation as the number of columns minus the number of rows.

1.2.2 The Fano index

Throughout the report we will also consider another invariant of Fano varieties, called the *Fano index*. Let us first recall here the precise definition of a Fano variety.

Definition 1.2.3. X is a *Fano variety* if it is a normal projective variety over the complex number with $\mathbb{Q}$-Cartier and ample anticanonical divisor. The anticanonical divisor is defined as $-K_X = \Lambda^n T_X$ (where T_X is tangent coherent sheaf).

Definition 1.2.4. The *Fano index* of a Fano variety X is the largest integer n such that K_X is divisible by n in $\text{Pic}(X)$. Note that if X is D-dimensional, then the index of X is less or equal to $D + 1$.

In the case of toric Fano varieties, the Fano index is the greatest common divisor of the sums of each row of a weight matrix giving rise to X.

Example 1.2.5. The Fano index of $\mathbb{P}^1 \times \mathbb{P}^1$ is two, since we can consider it as the GIT quotient specified by the weight matrix

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix},$$

whose columns sum to $\binom{2}{2}$.

1.3 Known classification results

In this section we summarise some known results in the classification of Fano varieties.

Smooth Fano varieties have been classified (up to deformation) in dimension one, two [DP87], and three [MM82, MM82]; see Table 1.1. If we restrict to smooth Fano varieties that are *toric*, the classification has been explored more, Table 1.3. There are eighteen examples in three dimensions [Bat81, Bat91, WW82] and 124 in four dimensions [Bat99, Sat00]. Smooth toric Fano varieties in dimension five were classified using an inductive algorithm by Kreuzer and Nill [KN09], and all the dimensions up to seven were classified by a different inductive algorithm due to Øbro [Øbr07].

Once we remove the smoothness requirement we still have some classification results. Isomorphism classes of weighted projective spaces with terminal singularities have been classified by Kasprzyk in dimensions up to four (inclusive); see Table 1.2 for a summary. Classifications in higher dimensions are hindered by the lack of an effective upper bound on a, the sum of the weights of a weighted projective space.

Isomorphism classes of toric Fano varieties have been classified by Kasprzyk up to dimension three (inclusive); see Table 1.3. Note that in dimension one and two having at worst terminal singularities and being smooth are equivalent. Classifying toric Fano varieties always relies on first classifying weighted projective spaces, so the lack of a classification in higher dimensions for the latter hinders the classification for the former.

Dimension		
1	2	3
$\mathbb{P}^1$	10	105
	see [DP87]	see [MM82, MM03]

Table 1.1: The known classification smooth Fano varieties in low dimension (up to deformation).

Dimension			
1	2	3	4
$\mathbb{P}^1$	$\mathbb{P}^2$	7	28 686
		see [Kas06]	see [Kas13]

Table 1.2: The known classification of terminal weighted projective spaces in low dimensions (up to isomorphism).

Dimension	Smooth	$\mathbb{Q}$-factorial and terminal	Total
1	1	1	1
2	5	5	5
3	18 see [Bat81, Bat91, WW82]	233 see [Kas06]	634 see [Kas06]
4	124 see [Bat99, Sat00]		
5	866 see [KN09]		
6	7 622 see [Øbr07]		
7	72 256 see [Øbr07]		

Table 1.3: The known classification of toric Fano varieties (up to isomorphism).

1.4 General assumptions

In this thesis, when we consider weighted projective spaces $\mathbb{P}(a_0, \ldots, a_N)$, we will always impose the following assumption.

Assumption 1.4.1. Given a weighted projective space $\mathbb{P}(a_0, \ldots, a_N)$ assume the following.

1. $a_i \in \mathbb{Z}_{>0}$ for all $i = 0, \ldots, N$.
2. $\gcd(a_0, \ldots, a_N) = 1$.
3. $\gcd(a_0, \ldots, \hat{a}_i, \ldots, a_N) = 1$, for all $i = 0, \ldots, N$.

Condition 3 assures that the set of singular points of the weighted projective space has dimension at most $N - 1$. It also means that weighted projective spaces respecting these conditions are uniquely identified by their weights. This condition is called *well-formedness*. All weighted projective spaces are $\mathbb{Q}$-factorial (since their fans have $D + 1$ rays in $\mathbb{R}^D$, so they are trivially simplicial), hence we do not need any extra assumption.

For the remainder of this thesis the weight matrices considered for toric Fano varieties of Picard rank two will always satisfy the following conditions.

Assumption 1.4.2. Given a weight matrix

$$\begin{bmatrix} a_1 & \cdots a_N \\ b_1 & \cdots b_N \end{bmatrix}$$

assume the following.

1. The columns of the matrix span a strictly convex cone in $\mathbb{R}^2$.
2. None of the columns are the zero vector.
3. The sum of the columns is not a multiple of any of them.

4. The subspaces S_+ and S_-, defined above, are both of dimension at least two.

5. The weight matrix is *well-formed*.

Conditions 1 and 2 together guarantee that the corresponding fan Σ is complete; that is, its support covers $\mathbb{R}^{N-2}$. The toric variety X is therefore projective (i.e. compact). Condition 3 ensures that each top-dimensional cone in the fan has $N-2$ rays; that is, the fan is simplicial. This implies that the toric variety X is $\mathbb{Q}$-factorial. Condition 4 ensures that each of the vectors $e_1, \ldots, e_N$ generates a one-dimensional cone $\mathbb{R}_{\geq 0} e_i$ in the fan Σ. Together with $\mathbb{Q}$-factoriality, this implies that the Picard rank of X is two and not lower.

Condition 5 is equivalent to condition 3 in Assumption 1.4.1. It requires the variety to have no *quotient grading*. This is a natural requirement since it identifies the simply connected algebraic varieties. Considering only well-formed weight matrices guarantees that a toric variety determines and is determined by its weight matrix, uniquely up to $\mathrm{GL}_2(\mathbb{Z})$ and S_N actions. Explicitly this is checked by ensuring that every submatrix obtained by removing one column is *standard*. Here, we define an $r \times N$ matrix to be standard if the greatest common divisor of its $r \times r$ minors is one.

1.5 The regularized quantum period

In this section we introduce precisely the ingredients we will need in the context of the study of mirror symmetry for Fano varieties.

1.5.1 Fano varieties

As mentioned in the Introduction, one of the central objects in our investigation is the regularized quantum period of a Fano variety.

Let X be a Fano variety. If X is smooth, then it is called a *Fano manifold*. We will not consider the general case of X not smooth, but only when we have nice well-behaved singularities. Namely, this is the case of X being an (N-)orbifold, meaning that X is locally analytically like $\mathbb{C}^N/G$ for a finite group G acting faithfully on $\mathbb{C}^N$. Given a Fano orbifold X, we can define n-pointed stable map of genus g as a holomorphic map

$$f : (\Sigma, x_1, \ldots, x_n) \to X \,,$$

where Σ is a curve of genus g and $x_1, \ldots, x_n$ are n marked smooth points on it. Such a map is *stable* if it has only nodes as singularities and has only finitely many automorphisms. We can define a moduli space of such maps as

$$X_{g,n,k} := \{ f \mid f \text{ is a } n\text{-pointed stable map of genus } g \text{ with } \deg f^*(-K_X) = k \} / \sim \,,$$

where $\sim$ is reparametrisation. We can define a power series, called the *quantum period* for X

$$G_X(t) = \sum_{d \geq 0} c_d t^d$$

where

$$c_d = \int_{[X_{0,1,d}]^{\text{vir}}} \psi^{d-2} \text{ev}^*(\text{pt}) \, .$$

Here, pt is the Poincaré dual to a point (i.e. the cohomology class defined by the volume form) and the evaluation map is defined as

$$\begin{aligned} \text{ev} : X_{0,1,d} &\to X \\ (\Sigma, f, x) &\mapsto f(x) \end{aligned}$$

and $\psi = c_1(L)$, where L is the line bundle on $X_{0,1,d}$ with fibre $L_f = T^*_x\Sigma$. We call the sequence of coefficients of the quantum period the *period sequence*.

In particular, we will consider the regularized version of the quantum period, which is defined as

$$\hat{G}_X(t) = \sum_{d \geq 0} c_d d! t^d$$

(and the corresponding *regularized period sequence* is defined accordingly). This is the Fourier–Laplace transform of $G_X(t)$.

Note that the regularized period sequence looks like an infinite amount of data, but it is not. In fact, there exists a differential operator [CCG+13, Theorem 4.3]

$$\hat{Q}_X(t) = \sum_{j=0}^{k} p_j(t) D^j \, ,$$

where $D = t\frac{d}{dt}$, $p_j \in \mathbb{Z}[t]$, such that $\hat{Q}_X \cdot \hat{G}_X \equiv 0$. This differential operator is defined up to multiplication by a constant if both k and $\deg p_k$ are as small as possible. The operator $\hat{Q}_X$ allows us to determine the coefficients of $\hat{G}_X$ from a finite amount of information, since it establishes a recursive relationship between the coefficients of the (regularized) period sequence.

Remark 1.5.1. When studying the regularized quantum period from the point of view of machine learning and data analysis in Chapter 3, we will only look at a truncated list of its coefficients. The existence of this differential operator $\hat{Q}_X(t)$ means that it is reasonable to expect to conclude general results (via machine learning) from a truncated version of the period sequence: we are not missing or throwing away information, since the entire structure is determined by its initial terms.

1.5.2 Laurent polynomials

In the context of mirror symmetry for Fano varieties, the mirror objects to Fano varieties are Laurent polynomials[7].

[7]The actual mirror objects are Landau-Ginzburg models (Y, w), where Y is an n-complex manifold, and $w : (Y, \Omega) \to \mathbb{C}$ is a holomorphic function, where Ω is a holomorphic n-form on Y. In our case we will always have Y to be a torus and w a Laurent polynomial

Definition 1.5.2. Given a Laurent polynomial $f \in \mathbb{C}[x_1^{\pm 1}, \ldots, x_n^{\pm 1}]$, its *classical period* is defined as

$$\pi_f(t) = \left(\frac{1}{2\pi i}\right)^n \int \frac{1}{1-tf} \frac{\mathrm{d}x_1}{x_1} \cdots \frac{\mathrm{d}x_n}{x_n} .$$

The classical period for a Laurent polynomial f can be written as power series

$$\pi_f(t) = \sum_{m=0}^{\infty} c_m t^m$$

where

$$c_m = \mathrm{coeff}_1(f^m) .$$

This can be seen by applying the Residue Theorem multiple times.

As for the regularized quantum period there exists a differential operator [CCG+13, Theorem 3.2]

$$L_f = \sum_{j=0}^{k} p_j(t) D^j$$

where $D = t\frac{d}{dt}$, $p_j \in \mathbb{Z}[t]$, such that $L \cdot \pi_f \equiv 0$. Again, this is unique up to multiplication by a constant if k and $\deg p_k$ are as small as possible. The recursion relations on the coefficients of the classical period are

$$\sum_{k \leq m} p_k(m-k) c_{m-k} = 0$$

for $m = 0, 1, 2, \ldots$.

Let us look at an example in the case of a well-known mirror of $\mathbb{P}^2$.

Example 1.5.3 (Example 3.5, [CCG+13])**.** Consider the Laurent polynomial $f = x + y + \frac{1}{xy}$. Its classical period is calculated as follows,

$$\begin{aligned}
\pi_f(t) &= \frac{1}{(2\pi i)^2} \int_{S^1 \times S^1} \frac{1}{1-tf} \frac{\mathrm{d}x}{x} \frac{\mathrm{d}y}{y} = \frac{1}{(2\pi i)^2} \int_{S^1 \times S^1} \sum_{k=0}^{\infty} t^k f^k \frac{\mathrm{d}x}{x} \frac{\mathrm{d}y}{y} \\
&= \sum_{k=0}^{\infty} t^k \frac{1}{(2\pi i)^2} \int_{S^1 \times S^1} f^k \frac{\mathrm{d}x}{x} \frac{\mathrm{d}y}{y} = \sum_{k=0}^{\infty} t^k \mathrm{coeff}_1(f^k) \\
&= \sum_{k=0}^{\infty} t^k \mathrm{coeff}_1 \left(\sum_{a+b+c=k} \frac{k!}{a!b!c!} \frac{x^a y^b}{(xy)^c} \right) = \sum_{k=0}^{\infty} \frac{(3k)!}{(k!)^3} t^{3k} .
\end{aligned}$$

We note that the coefficients satisfy the following recursive relation

$$k^2 c_{3k} - 3(3k-1)(3k-2) c_{3k-3} = 0$$

which is equivalent to the differential operator

$$[D^2 - 27t^3(D+1)(D+2)]\pi_f \equiv 0 .$$

1.5.3 Mirrors

Definition 1.5.4. Given a Fano variety X and a Laurent polynomial f, we say that they are *mirror partners* if $\hat{G}_X \equiv \pi_f$.

It is conjectured that there is a bijection between deformation classes of certain Fano varieties and mutation classes of certain Laurent polynomials[8], under which the regularized quantum period corresponds to the classical period; see [CKPT21]. Therefore, this leads to the related expectation that the regularized quantum period is a complete invariant of Fano varieties.

1.5.4 Toric varieties and toric complete intersections

Gromov–Witten invariants are notoriously hard to compute. Fortunately, for certain classes of Fano varieties, namely when they are toric and toric complete intersections, we have some explicit formulae to compute the regularized quantum period.

For a weighted projective space $X = \mathbb{P}(a_0, \ldots, a_N)$, the regularized quantum period is computed using the formula [CCGK16],

$$\hat{G}_X(t) = \sum_{k \in \mathbb{N}} \frac{(ka)!}{\prod_{i=0}^{N} (ka_i)!} t^{ka} \tag{1.5.1}$$

where $a = \sum_{i=0}^{N} a_i$.

In general, assume we are given a toric variety X by an $r \times N$ weight matrix, as in Section 1.1. Let $(\alpha_i)_{i=1,\ldots,N}$ be the columns of the weight matrix. Then the regularized quantum period is given by the formula [CCGK16],

$$\hat{G}_X(t) = \sum_{k \in \mathbb{Z}^r \cap C} \frac{(\alpha^T \cdot k)}{\prod_{i=1}^{N} (\alpha_i^T \cdot k)!} t^{\alpha \cdot k}\,, \tag{1.5.2}$$

where $\alpha = \alpha_1 + \cdots + \alpha_N$ and C is the convex cone

$$C = \{x \in \mathbb{R}^r \mid \alpha_i^T \cdot x \geq 0 \text{ for all } i = 1, \ldots, N\}\,.$$

Note that in [CCGK16] the above is stated only for smooth varieties. However, this is generalised to our case since we are considering projective $\mathbb{Q}$-factorial varieties; [CCIT15].

Example 1.5.5. Consider projective space $\mathbb{P}^2$, which is a smooth toric variety with weight data $(1, 1, 1)$. Using (1.5.1) we obtain that its quantum period is

$$G_{\mathbb{P}^2}(t) = \sum_{k \geq 0} \frac{1}{(k!)^3} t^{3k}\,,$$

[8] These are MMLP (maximally mutable Laurent polynomials); see [CKPT21] for the definition. Mutations of Laurent polynomials can be understood combinatorially in terms of rearrangement of slices of the corresponding Newton polytope (i.e. the polytope obtained as the convex hull of the exponents in the Laurent polynomial).

and the regularized version is

$$\hat{G}_{\mathbb{P}^2}(t) = \sum_{k\geq 0} \frac{(3k)!}{(k!)^3} t^{3k} \,. \tag{1.5.3}$$

Note that this coincides with the classical period for the Laurent polynomial in Example 1.5.3, since $\mathbb{P}^2$ is its mirror.

Lastly, consider a smooth $\mathbb{Q}$-factorial toric variety X, whose data is given by an $r \times N$ integer valued matrix with columns $(\alpha_i)_{i=1,\dots,N}$. Consider the smooth Fano complete intersection Y in X defined by a section of $E = L_1 \oplus \cdots \oplus L_M$, for line bundles L_i. Let $\beta_i = c_1(L_i)$, i.e. the first Chern classes of the line bundles[9]. This data can also be written in an $r \times M$ matrix with columns $(\beta_i)_{i=1,\dots,M}$. Then the quantum period of Y is [CCGK16, Corollary D.5]

$$G_Y(t) = \sum_{k\in\mathbb{Z}^r\cap C} \frac{\prod_{i=1}^M (\beta_i^T \cdot k)!}{\prod_{i=1}^N (\alpha_i^T \cdot k)!} (\omega^T \cdot k) t^{\omega^T \cdot k} \,, \tag{1.5.4}$$

where $\omega = \sum_{i=1}^N \alpha_i - \sum_{i=1}^M \beta_i$ and C is the convex cone

$$C = \{x \in \mathbb{R}^r \mid \alpha_i^T \cdot x \geq 0 \text{ for all } i = 1, \dots, N\} \,.$$

If $c_d = d! \sum_{\omega^T\cdot k = d} \frac{\prod_{i=1}^M (\beta_i^T\cdot k)!}{\prod_{i=1}^N (\alpha_i^T\cdot k)!}$, then the regularized version of $G_Y(t)$ is

$$\hat{G}_Y(t) = \sum_{d\geq 0} \sum_{k=0}^{d} \binom{d}{k} (-c_1)^{d-k} c_k t^d \,. \tag{1.5.5}$$

This formula holds outside the smooth case, for quasi-smooth[10] (a generalisation of smoothness) toric complete intersections in a $\mathbb{Q}$-factorial toric ambient space, which have an orbifold[11] structure and when the line bundles L_i's are convex[12] [Wan19, SW22].

Example 1.5.6. Let Y be a smooth cubic threefold: it is the hypersurface of degree three in $\mathbb{P}^5$. The toric ambient weight data is $(1, 1, 1, 1, 1)$, and $\beta = 3$. Therefore, $\omega = 2$ and its quantum period is

$$G_Y(t) = \sum_{k\geq 0} \frac{(3k)!}{(k!)^5} t^{2k} \,,$$

and its regularized quantum period is

$$\hat{G}_Y(t) = \sum_{k\geq 0} \frac{(2k)!(3k)!}{(k!)^5} t^{2k} \,.$$

[9] This is an element of $H^2(X;\mathbb{Z})$.

[10] Let Y be defined by the complete intersection of polynomials $f_1, \dots, f_M$ in the toric variety X, for $f_i \in H^0(X; L_i)$. Recall that the toric variety is given as the quotient of $U_\omega/(\mathbb{C}^\times)^r$ as discussed before. Then Y is smooth if the common zero set of $f_1, \dots, f_n$ in U is a smooth subvariety of codimension n.

[11] Recall that an orbifold is a space that locally looks like $\mathbb{C}^n/G$ for G a finite group. More precisely, we are requiring the toric complete intersection to be a smooth Deligne–Mumford stack with projective coarse moduli space.

[12] This is a positivity assumption on the line bundles. For example every ample line bundle is convex.

2 Machine Learning Background

In the first part of this chapter we introduce the basic terminology used in the context of machine learning problems. This is aimed at non-specialists and does not survey machine learning in its entirety. Instead, we introduce only those algorithms and tools that will be used in Chapters 3 and 5. We point the reader to the references for a more comprehensive introduction [GBC16, RS20, Gér22]. In the second part of this chapter, we survey various applications of machine learning to pure mathematics. For discussions of a similar flavour see [Wil23, He23].

2.1 Machine learning fundamentals

Machine learning is a subset of artificial intelligence that refers to a suite of algorithms that aims at processing data and learning patterns within the data improving accuracy over time. Machine learning algorithms are (historically) classified into three types: supervised, unsupervised, and reinforcement learning.

- *Supervised learning* refers to those algorithms that aim at predicting a label or a quantity given some features of a datapoint. Here, the labels or the property that we aim to predict are included into the data that is shown to the algorithm during the training step (this is called *training data*). Therefore, for supervised learning we must have a correctly labelled dataset. A classic example of supervised learning is spam filters.
- *Unsupervised learning* refers to algorithms that do not get trained on the labels of data. Instead, they are trained on data without labels and find patterns within the data without a specific instruction. Examples of this are clustering algorithms, like in market segmentation or social network analysis.
- Lastly, the term *reinforcement learning* is used to describe those algorithms that are designed as a game. It consists of an agent which observes its environment, select and performs actions, and gets feedback via a reward function. The result is the agent learning the best strategy to maximise the reward function. Perhaps, the most famous example is AlphaGo [SHM+16].

In this thesis, we will only apply machine learning algorithms in a supervised fashion. Let us outline the workflow of using machine learning on a dataset, before going into the details of individual algorithms.

Data in this thesis will always have a vector of features and a label associated to it, looking something like Table 2.1. Not all data (of course) has this shape, for example we could have data consisting of matrices (in the case of image classification tasks, for example) or graphs (a popular representation for problems relating to social networks data). However, in our context our datasets will always have a vector representation.

Our aim is to build a model that takes in the features of a certain sample and returns a label as accurately as possible. In particular, since we will work within a *supervised learning* framework, we will train the machine learning model using the known labels to optimise the model parameters. This workflow can be broken down into the following steps:

- *Model selection*. Given a problem the user chooses a machine learning model that is most suitable.
- *Train–test split*. The data is then split into training (used to train the algorithm), and testing (used to evaluate the algorithm at the end of training). We will refer to the percentage of data used for training as the train–test split.
- *Data preprocessing*. Depending on the choice of model, the data might need to undergo some preprocessing step, for example some scaling and normalisation.
- *Hyperparameter tuning*. A small amount of training data is held out for tuning the hyperparameters of the models. Most machine learning models come with hyperparameters that can be chosen by the user, and their value can affect training time and accuracy greatly. By trying out different hyperparameters, the user can decide on the most promising configuration to fix.
- *Training the model*. The model, with specified hyperparameters, is trained using the training data. Note that since we are doing supervised learning, the model is shown the correct labels of the training samples during this step.
- *Testing the model*. The model is evaluated on the testing data, which was completely unseen during training. In the case of classification, the model is evaluated using *accuracy*, which is the percentage of testing samples that are correctly labelled by the model.

	Feature 1	$\cdots$	**Feature** k	**Label**
sample 1	x_1^1	$\cdots$	x_k^1	y^1
sample 2	x_1^2	$\cdots$	x_k^2	y^2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
sample N	x_1^N	$\cdots$	x_k^N	y^N

Table 2.1: Data with N samples, each of it has k features and a label.

2.2 Preprocessing steps

2.2.1 Feature scaling

With a few exceptions, most machine learning model perform better when the features are scaled. There are two common approaches to feature scaling: *min-max scaling* and *standardisation*.

Assume we have been given a dataset with N samples each of which has a feature x_i^j for $j = 1, \ldots, N$. Min-max scaling (often called normalisation) scales each feature x_i^j in the following way

$$\text{MinMax}(x_i^j) = \frac{x_i^j - \min_k(x_i^k)}{\max_k(x_i^k) - \min_k(x_i^k)} \, .$$

This results in all the features belong to the range $[0, 1]$.

On the other hand standardisation scales each x_i^j so that the resulting distribution has zero mean and unit variance, i.e.

$$\text{StandardScaler}(x_i^j) = \frac{x_i^j - \sum_{k=1}^{N} x_i^k}{\sigma}$$

where σ is the standard deviation. Unlike the min-max scaling, standardisation does not bound the feature values and it is much less affected by outliers.

2.2.2 Principal component analysis

Dimensionality reduction is a very important technique used in machine learning problems. In fact, datasets often include thousands of features which slow down the training regime. Moreover, datasets with many features are often representative of very sparse point clouds in high-dimensional space, meaning that there is the need to increase the number of training samples to assure that the machine learning model is learning on a representative sample of the distribution. This is commonly referred to as the *curse of dimensionality*. In these cases, it is useful to employ techniques that reduce the dimensionality of the data, whilst retaining as much information as possible. Another reason dimensionality reduction is important is to aid in visualisation by reducing the number of input features to two or three, which can then be plotted.

A common technique to reduce the dimensionality of the data is projecting to a hyperplane of a lower dimension. In choosing the hyperplane we would like to be able to determine the one that contains the *most* amount of information possible. This is the aim of Principal Component Analysis (PCA). PCA identifies new axes (called *principal components*) along which the data varies the most: these are orthogonal to each other, so that they are uncorrelated.

Let $X \in \mathbb{R}^{N \times k}$ be a matrix containing the feature values for all the (N) samples. The first

step is to compute the covariance matrix

$$\text{Cov} = \frac{1}{k-1}(X - \bar{X})(X - \bar{X})^T$$

where $\bar{X}$ is the matrix of column-wise means (this is because PCA assumes that our data is centred at zero, hence we need to translate the mean to zero). Note that Cov is a symmetric matrix, so it can be diagonalised

$$\text{Cov} = VLV^T$$

where V is the matrix of eigenvectors, and L is the diagonal matrix of eigenvalues. Now, if we want to perform PCA with d principal components (meaning that we are projecting to a hyperplane of codimension $(N - d)$), we consider V_d which is V where we are selecting only the d eigenvectors with largest d eigenvalues. Then, the projection of the data along the top d principal components is

$$\text{PCA}(X, d) = XV_d \in \mathbb{R}^{N \times d}\,.$$

We refer to the normalised eigenvalue corresponding to each component of the PCA as its *explained variance*. It intuitively measures how much of the total variance of the original dataset is explained by each PCA component.

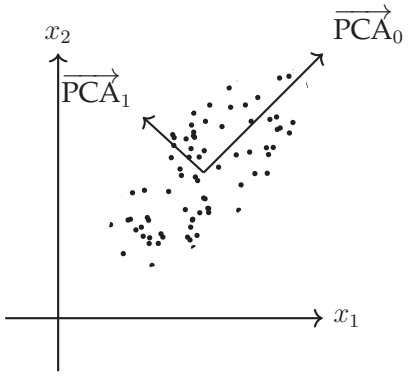


Figure 2.1: Pictorial representation of PCA with two components applied on two-dimensional data.

2.3 Some models

2.3.1 Linear regression

Linear regression is an example of a regression model (i.e. the dataset labels are continuous rather than discrete, as they would be for a classification model) that takes in some input features $\mathbf{x} \in \mathbb{R}^k$, with label $y \in \mathbb{R}$, and predicts the value of a dependent variable $\hat{y} \in \mathbb{R}$ as

$$\hat{y} = \mathbf{w}^T \cdot \mathbf{x} + b\,, \tag{2.3.1}$$

for $\mathbf{w} \in \mathbb{R}^k$ and $b \in \mathbb{R}$. A linear regression model is trained by finding the values of $\mathbf{w}$ and b such that the linear output $\hat{y}$ best approximates the actual label y. This is done by

optimising the parameters $\mathbf{w}$ and b to minimise some measure of the error between the predicted and the actual values. For example, the *Mean Squared Error* (MSE) is commonly used and it is defined as

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.3.2}$$

where N is the number of training samples. The result of running linear regression is represented pictorially in Figure 2.2.

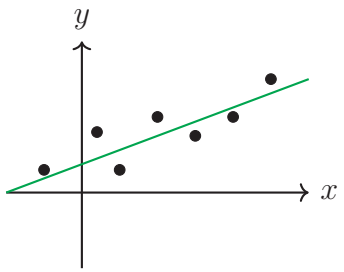


Figure 2.2: Pictorial representation of linear regression. The black dots are data points $\{(x_i, y_i)\}_i$ and the green line is the line of best fit.

When applying linear regression to data with just one feature (i.e. $x_i \in \mathbb{R}$ for $i = 1, \ldots, N$) we can compute the *standard error* of the two coefficients that determine the line of best fit, i.e. the slope and the y-intercept ($w, b \in \mathbb{R}$ in (2.3.1)). The standard error helps us determine how representative the sample data is of the larger population.

Let $(x_1, \ldots, x_N)$ be the features and $(y_1, \ldots, y_N)$ the targets for N samples. Let $\bar{x} = \sum_{i=1}^{N} x_i$. Then, we define the standard error of the slope of the linear regression as

$$s_{\text{slope}} = \left(\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{N-2} \times \frac{1}{\sum_{i=1}^{N}(x_i - \bar{x})}\right)^{1/2} \tag{2.3.3}$$

and the standard error of the y-intercept as

$$s_{\text{int}} = \left(\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{N-2} \times \frac{N\bar{x}^2 + \sum_{i=1}^{N}(x_i - \bar{x})^2}{N\sum_{i=1}^{N}(x_i - \bar{x})}\right)^{1/2}. \tag{2.3.4}$$

The standard error allows us to assess the accuracy of the predictions. Approximately 95% of the observations should fall within an absolute distance of twice the standard error from the regression line. Therefore, a low standard error suggests that the datapoints all lie close to the regression line, while a high standard error means that the datapoints are further away from the line of best fit.

2.3.2 (Linear) Support vector machines

In this thesis we will make use of a Support Vector Machine (SVM) with linear kernel for classification. Given a labelled point cloud of data belonging to two classes, a linear SVM computes the optimal hyperplane separating the two classes.

In practice, a linear SVM relies on a decision function, which takes in the features $\mathbf{x} \in \mathbb{R}^k$ of a sample and assigns it a label (in this case ± 1, which correspond to the two classes)

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^T \cdot \mathbf{x} + b \geq 0 \\ -1 & \text{if } \mathbf{w}^T \cdot \mathbf{x} + b < 0 \end{cases} .$$

Here $\mathbf{w} \in \mathbb{R}^k$ and $b \in \mathbb{R}$ are the parameters defining the hyperplane in $\mathbb{R}^k$, also called *decision boundary*. The problem of finding the *optimal* hyperplane dividing the two classes of data points is equivalent to maximising the margins around the hyperplane. The margins are defined as the distance between the hyperplane and the closest two data points belonging to the two classes. These data points are called *support vectors*. A pictorial representation of the hyperplane resulting from a linear SVM is shown in Figure 2.3.

Note that since $\mathbf{w}^T \cdot \mathbf{x} + b = 0$ and $c(\mathbf{w}^T \cdot \mathbf{x} + b) = 0$ define the same hyperplane (for any $c \neq 0$), then we are able to normalise $\mathbf{w}$ such that $\mathbf{w}^T \cdot \mathbf{x_1} + b = 1$ and $\mathbf{w}^T \cdot \mathbf{x_2} + b = -1$, for $\mathbf{x_1}, \mathbf{x_2}$ support vectors. Therefore, the margin of the hyperplane is defined as

$$\frac{\mathbf{w}^T \cdot \mathbf{x_1} + b - \mathbf{w}^T \cdot \mathbf{x_2} - b}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|} ,$$

where $\|\mathbf{w}\|$ is the norm of $\mathbf{w}$. Therefore, we are trying to solve the constrained optimisation problem

$$\begin{aligned} &\min_{\mathbf{w}}(\|\mathbf{w}\|^2) \text{ subject to} \\ &y_i(\mathbf{w}^T \cdot \mathbf{x_i} + b) > 1 \text{ for all } i = 1, \ldots, N . \end{aligned}$$

Note that we optimise with respect to $\|\mathbf{w}\|^2$ instead of $\|\mathbf{w}\|$ since the latter is not differentiable at the origin.

The above problem solves the *hard-margins* linear SVM objective, since it does not allow for outliers. However, this does not work well in practice since it assumes that the data is linearly separable: in real-world applications this is often not the case. Therefore, in practice we need a more flexible model, which is the *soft-margin* version of linear SVM. This is done by introducing a regularisation parameter C: a lower value of C allows more flexibility with outliers, while a higher value of C is stricter. Note that C is an example of *hyperparameter* (which we have mentioned in the previous section) that needs to be fixed before training the model.

Explicitly, for the soft margin SVM we introduce a *slack* variable $\mathbf{z} = (z_1, \ldots, z_N) \in \mathbb{R}^N_{\geq 0}$ and

solve the following optimisation problem

$$\min_{\mathbf{w},z_i} \left(\|\mathbf{w}\|^2 + C \sum_{i=1}^{N} z_i \right) \text{ subject to}$$
$$y_i(\mathbf{w}^T \cdot \mathbf{x_i} + b) > 1 - z_i \text{ for all } i = 1, \ldots, N \, .$$

Then, the regularisation parameter C tells us how much weight to give to the slack variables.

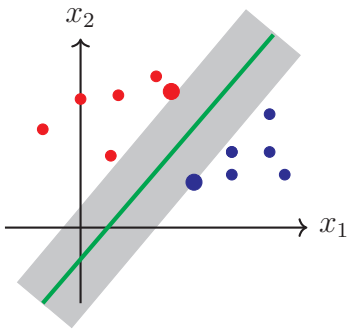


Figure 2.3: Pictorial representation of a linear SVM on two-dimensional data. The green line is the decision boundary and the colours of the dots (red and blue) correspond to datapoints with two different labels. The grey area represents the margins of the decision boundary. The bigger dots are two support vectors.

Note that as we have presented linear SVM here it seems to only support binary classification. This is indeed the case, however we can tackle multi-class classification problems using multiple support vector machines. This is done by breaking down the problem into individual binary classification problems, one for each pair of classes.

2.3.3 Artificial neural networks

An Artificial Neural Network (ANN) is a type of machine learning architecture. It can be used in the context of supervised, unsupervised, and reinforcement learning. In this section, for expository purposes, let us consider the supervised learning classification problem of detecting handwritten digits to go through the details of this algorithm; see Figure 2.4 for some examples of labelled data points from the MNIST dataset [LCB10].

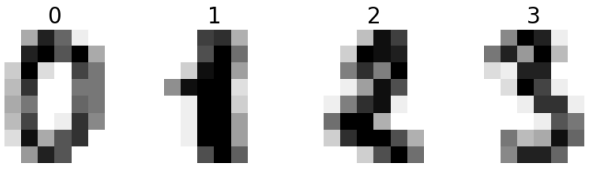


Figure 2.4: Examples of 8×8 images of handwritten digits and their labels, from the MNIST dataset [LCB10].

The architecture of a neural network is represented by a multipartite graph, where the first layer takes in the input values (the features), and the last layer gives a prediction of the

correct label. The nodes of the graph are called *neurons* and the edges are called *connections*. Assume we are given the MNIST dataset [LCB10] consisting of 8×8 grey-scale images of handwritten digits (from 0 to 9), which have been correctly labelled. The supervised learning task that we aim to solve is to train a machine learning algorithm to predict the correct digit label given the pixel values of an image. Therefore, our neural network will have an input layer made of 64 neurons (since we have $8 \times 8 = 64$ pixels in each image), taking the input values of the pixels of an image, and an output layer made of 10 neurons, one for each possible class (since we are classifying digits between 0 and 9). The output of the network will be a probability distribution $(p_0, \ldots, p_9)$, where p_i will correspond to how *probable* it is for the label of the image to be i. Therefore, the desired output for an image with label 0 is $(1, 0, \ldots, 0)$; see Figure 2.5.

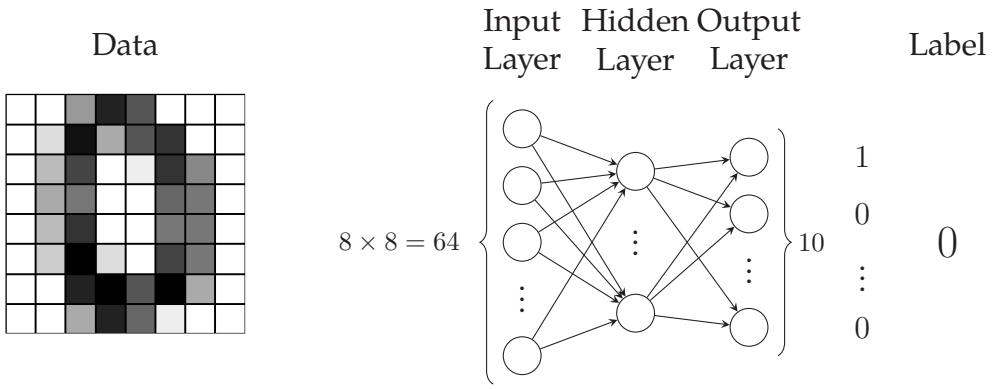


Figure 2.5: Schematic representation of a neural network with one hidden layer predicting labels for pictures of handwritten digits.

Between the input and the output layers there are hidden-layers, which are also made out of neurons. Between two consecutive layers in a neural network, we have edges between all vertices. The edges represent affine-linear maps. For example if we have three neurons in a layer (a_1, a_2, a_3) connected to a neuron in the following layer (as shown in Figure 2.6), then the value of neuron v_1 is

$$v_1 = a_1 w_1 + a_2 w_2 + a_3 w_3 + b$$

for some weights $w = (w_1, w_2, w_3) \in \mathbb{R}^3$ and a bias $b \in \mathbb{R}$. Each edge in a neural network correspond to a weight, and each neuron also carries a bias. These are all *learnable* parameters: they are initialised randomly, but will change during the training of the neural network, in order to optimise its output to be as close as possible to the correct label for as many datapoints as possible.

However, if we only had affine-linear maps between neurons we would only be able to use a neural network to approximate affine-linear functions. To counteract this, we add a non-linear element, called an *activation function*. An activation function is just a non-linear function that is composed with the affine-linear map. In the context of Figure 2.6, we would have

$$v_1 = \sigma(a_1 w_1 + a_2 w_2 + a_3 w_3 + b)$$

for some activation $\sigma : \mathbb{R} \to \mathbb{R}$. There are many activation functions to choose from, we recall a couple (ReLU and LeakyReLU) in the following example.

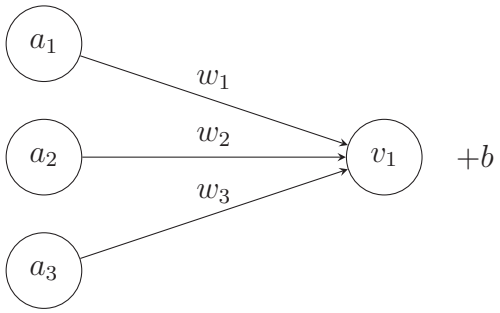


Figure 2.6: Example of connections to a neuron in a neural network, from three neurons in a previous layer.

Example 2.3.1. A popular activation function is the *rectified linear unit* (ReLU) [Aga18]. It is defined as $\sigma : \mathbb{R} \to \mathbb{R}$,

$$\sigma(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases} .$$

A variation of this is LeakyReLU, which includes a small positive slope for the negative values in order to mitigate the vanishing gradient problem [MHN13]. For a specified slope $s > 0$, we define LeakyReLU as $\sigma : \mathbb{R} \to \mathbb{R}$

$$\sigma(x) = \begin{cases} x & \text{if } x \geq 0 \\ s \cdot x & \text{otherwise} \end{cases} .$$

In summary, a neural network is just a composition of affine-linear maps (A_i) and activation functions (σ)

$$\mathbb{R}^{n_I} \xrightarrow{A_1} \mathbb{R}^{n_{H_1}} \xrightarrow{\sigma} \mathbb{R}^{n_{H_1}} \xrightarrow{A_2} \mathbb{R}^{n_{H_2}} \xrightarrow{\sigma} \cdots \mathbb{R}^{n_{H_k}} \xrightarrow{A_k} \mathbb{R}^{n_O} \tag{2.3.5}$$

where n_I is the number of inputs, n_O is the number of outputs, and n_{H_i} is the number of neurons in the ith hidden layer.

Note that in (2.3.5) we are using the same activation function throughout the network, except in the output layer where it is omitted. In the context of multi-classification (like our running example about classifying handwritten digits) the output layer should represent a probability distribution. Hence, the output is passed through the softmax $\sigma : \mathbb{R}^{n_O} \to \mathbb{R}^{n_O}$,

$$\sigma(x)_i = \frac{e^{x_i}}{\sum_{i=j}^{n_O} e^{x_j}} . \tag{2.3.6}$$

Now that we have the architecture of a neural network, we can address what it means to train a neural network. The parameters (i.e. the weights and biases of the affine-linear maps A_i's) are initialised randomly. We can feed our randomly initialised neural network function some data points (in our case images of handwritten digits) and see what the output is: since the network has not been trained at all, it will almost always produce the incorrect output. We can calculate the *cost* function (or *loss* function) between the correct output and the output of the neural network. There are many loss functions to choose from, for simplicity the reader can think of the *mean squared error* from (2.3.2), but we recall the usual loss function used in classification problems in the following example.

Example 2.3.2. Consider the binary classification problem of deciding whether a handwritten digit is 0 or 1. For N samples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, the output of the network will be a probability p_i of the label being 1, and probability $1 - p_i$ of the label not being 0 (for the ith sample). We compute the binary cross entropy loss as

$$-\frac{1}{N}\sum_{i=1}^{N} y_i \log(p_i) + (1 - y_i)\log(1 - p_i)\,.$$

This can be easily generalised to include multiple classes (not just 1 and 0) for multi-classification problems (then it is called just the *cross entropy loss*).

Once we have the loss function, we can perform gradient descent on the loss function to update the learnable parameters $\mathbf{w}$,

$$\mathbf{w} = \mathbf{w} - \alpha \nabla L(\mathbf{w})$$

where α is the *learning rate*. The learning rate is a quantity that regulates how big of a step we are taking when updating the weights: a larger learning rate means a model that changes more quickly but might not converge to the minima because the steps are too big. A smaller learning rate is less likely going to miss a minimum, but it is much slower, and might get stuck in the wrong local minimum unable to get out. The process of computing the gradient of the loss function is actually very slow and hard, so in practice algorithms use gradient estimation methods (i.e. backpropagation) and more sophisticated optimisers than naïve gradient descent (like stochastic gradient descent or adaptive gradient algorithms).

The process is repeated, and the learnable parameters keep being updated, until the loss function has found a satisfactory minimum (we would hope it is a global minimum, however since the loss landscape is usually a space that is very complicated and very far from being convex, this is usually a local minimum). Once we have reached a plateau of the loss function, we stop the training and test our network on completely unseen data – the testing data.

As an aside, let us remark on the difference between *parameters* and *hyperparameters*, in the context of neural networks. The parameters are the weights and biases: they specify the neural network function, and they are updated during training. On the other hand, a network depends on a number of hyperparameters, defining its architecture and the training regime, which are fixed before training (in the *hyperparameter tuning* portion of the machine learning workflow that we have described at the beginning of this section). Examples of hyperparameters in the context of neural networks are as follows.

- The size of the network (i.e. number of layers, number of neurons).
- The choice of loss function.
- The choice of activation function.
- The choice of optimisation algorithm and related optimisation parameters.
- The choice of learning rate or, in case we want the learning rate to change during training (for example, we might want it to decay), then the choice of learning rate scheduler.

The architecture as we have defined it describes a *feedforward fully-connected multi-layer perceptron*, which we will abbreviate as MLP. Feedforward means that the nodes in our architecture do not form loops. This is not always the case, for example recurrent neural networks are an example of neural networks which are not feedforward. Fully-connected refers to the fact that all connections layer to layer are present: each input node in our architecture is connected to each output node. Again this does not have to be the case: for example convolutional layers in convolutional neural networks are not fully-connected.

2.4 Performance measures and explainability

After we have trained a model, we need to evaluate its performance. Let us recall some performance measures.

Accuracy In the context of classification problems, to test the performance of a model we compute its accuracy. The accuracy is calculated only on testing data (data that has not been seen during training), and it is

$$\text{accuracy} = \frac{\#\text{correctly labelled samples}}{\#\text{all samples}} .$$

Confusion matrices Whilst accuracy is a compact performance metric, it is potentially misguided. In fact, if a model is biased and always gets wrong one class, this will not be obvious just by looking at the accuracy. Therefore, we use *confusion matrices* to give a better measure of the performance of a model. Confusion matrices are $n \times n$ matrices where n is the number of distinct classes in the problem. The rows correspond to the *actual classes* and the columns to the *predicted classes*. Therefore, the entry in the ith row and jth column will record how many samples with correct label i, have been misclassified as j. In Table 2.2 we give an Example for a hypothetical binary classifier with labels *Positive* and *Negative*. The accuracy of such a classifier would be 66.7%, however it is clearly not a good model, since it misclassifies the majority of samples with label Positive.

	Negative	**Positive**
Negative	15	10
Positive	0	5

Table 2.2: Example of confusion matrix for a binary classification problem with labels *Positive* and *Negative*.

Throughout this thesis, we often produce confusion matrices normalising the entries with respect to either of the axes: this is just to aid in visualisation.

Saliency analysis Saliency analysis is tool to aid in understanding why a neural network is performing well. It measures which inputs of the network influence the output the most. In this thesis, we will use saliency analysis in the form of SHapley Additive exPlanations

Values (SHAP). They were first introduced in the context of game theory to calculate the payout for each player depending on their contribution to the total payout [Sha53]. In the context of machine learning, they calculate the contribution of each feature to the overall prediction for a sample, compared to the average prediction for the data.

2.5 Machine learning for pure mathematics

We conclude this chapter by surveying works applying machine learning methodology to pure mathematics problems and objects. Part of this section covers content from [Ven24a].

One of the main aims of this thesis is to showcase the power of a workflow integrating machine learning, data analysis, and other computational techniques in algebraic geometry. However, this approach is potentially widely applicable to many areas of pure mathematics. Such a workflow can be summarised in a sequence of steps, ensuring a mathematical question is amenable to these methods, gathering the correct data, and building the machine learning model; see Figure 2.7. Let us highlight the considerations one must keep when walking through the steps of this workflow.

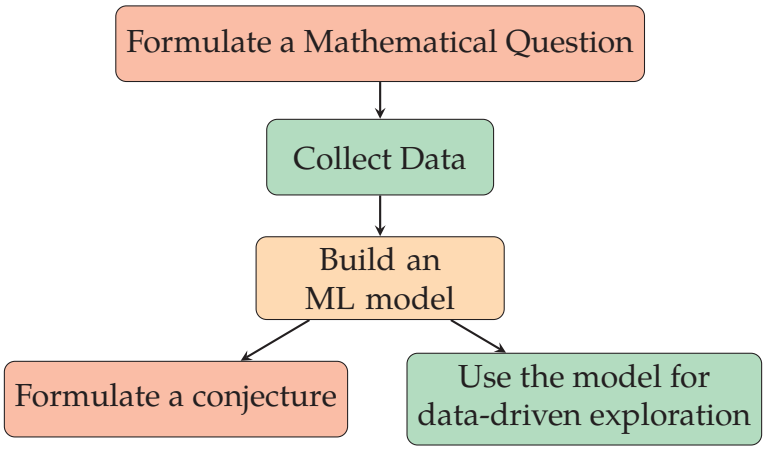


Figure 2.7: The steps of applying machine learning to pure mathematics problems: red are mathematical steps, green are computational steps, and yellow is a machine learning step.

The workflow starts with the formulation of a mathematical question that is amenable to this data-driven methodology, for example we could ask if a certain property can be predicted by machine learning or whether it is able to understand an unknown relationship between different mathematical invariants. The key property of such a question is that we need to be able to generate many distinct examples to train on, in a relatively short amount of time. This workflow is particularly effective when applied to those problems for which we have an abundance of data, but lack a general rigorous understanding. The datasets needed could be already available (examples of big mathematical datasets were mentioned in the Introduction) or might need to be generated from scratch. The problem of generating a large enough dataset might mean that the question needs to change, for example we might need to restrict to a class for which there are algorithms that can be used to generate samples (an example of this is how, in this thesis, we always restrict to the case of toric Fano varieties). One must bear in mind that the data generation process can

introduce bias in different ways. Is the class of objects appearing in the data representative of a more general class? Will the sampling methodology influence the results? Questions of this type should be considered, and they are the reason why the data generation step is of fundamental importance in this workflow.

After the data has been generated, or obtained from pre-existing datasets, the next step is the choice of the model. There are many frameworks to choose from. An important note to consider when choosing a model is the balance between explainability and performance. A model like a Support Vector Machine (see Section 2.3.2) is very easy to decipher, but might not perform as well as a Neural Network (see Section 2.3.3). However, the latter is much harder to interpret! Being able to build a machine learning model that can relate mathematical quantities when there is no theoretical understanding why they should be related, is already important when guiding intuition: the confidence that a statement must be true can guide centuries of research (consider for example Fermat's Last Theorem). However, any explainability we can get from a model can be a huge help when trying to come up with the appropriate mathematical statement and proof. There have been examples of using saliency analysis to understand the importance of different features in the predictions made by a model [DVB+21, JCB+23] – and we will bring an example of this in Chapter 3.

Once we have built a high-accuracy machine learning model that answers our question there are many ways that it can be used to drive mathematical discovery. As mentioned in the Introduction, in this thesis we will explore two ways in which mathematicians can use trained machine learning models: aiding theorem conjectures, and replacing expensive computational routines. We note that the first example of applications of a machine learning driven workflow to inspire conjectures in pure mathematics is [DVB+21]. In this work, which was a collaboration of machine learning researcher from DeepMind and mathematicians from the University of Oxford and the University of Sydney, the authors use machine learning and saliency analysis to guide conjecture generation in knot theory and representation theory, which are subsequently proven using traditional methods [BBD+22, DJLT21]. There have been more examples of machine learning applied to mathematical structures, to predict quantities that are hard to compute, in the context of string theory [BHJM18, DLQ22], knot theory [GHMR23, Hug20], combinatorics [CHK23, BHH+23, JCB+23], algebraic geometry [HKS22, BHH+22], and number theory [HLOP22, Lee23, HLO22].

Finally, whilst we will concentrate on applications of machine learning in this flavour – for guiding conjecture generation – let us remark that these are not the only examples of interactions between pure mathematics and machine learning. Machine learning routines have been used to construct examples of mathematical objects [BHH+24], or counterexamples to conjectures [Wag21]. On the other hand, there are many examples of machine learning pipelines being used to accelerate exact computational routines: in the context of Gröbner basis calculations [PSHL20, MPP23, KIK+23, WDL22], computer algebra [HEW+14, HKS22], symbolic mathematics [LC20]. Lastly, all approaches we have mentioned are complemented by works in using Large Language Models (LLMs) to guide mathematical proofs and mathematical reasoning [RPBN+24, IDS23, AAt24, PHZ+22]. This is often done in conjunction with formal proof assistant, such as LEAN [YSG+24], Isabelle [JLHW21]. We note that in contrast with this work, our methodology often employs classical machine learning architectures, which are relatively low in computational cost compared to LLMs, and therefore are more accessible to the working mathematician.

3 The Dimension of a Fano Variety

The aim of this chapter is to investigate whether the regularized quantum period of a Fano variety contains geometric information about the variety. Recall that the regularized quantum period of a Fano variety is a power series

$$\hat{G}_X(t) = \sum_{d=0}^{\infty} c_d t^d \,,$$

where $c_0 = 1$, $c_1 = 1$, and $c_d = r_d d!$, where each r_d are certain Gromov–Witten invariants of X. We will refer to the coefficients $(c_d)_d$ as the *period sequence* of X. It is conjectured that the regularized quantum period is a complete invariant of Fano varieties. Therefore, we should be able to recover all the geometric information about X given its period sequence. To explore this we look at one of the easiest invariants: does the regularized quantum period know the *dimension* of the variety? We investigate this question using machine learning tools on two datasets, one of weighted projective spaces and one of Picard rank 2 toric varieties. In both cases we require that are varieties are toric $\mathbb{Q}$-Fano.

This chapter covers some of the content of [CKV23b].

3.1 Data generation

In this section we detail the data generation steps for the weighted projective space dataset (`DSdim_wps`) and for the toric varieties of Picard rank two (`DSdim_rk2`). The datasets are generated using Magma v2.25-4, [BCP97].

3.1.1 Weighted projective spaces

The dataset `DSdim_wps` consists of $150\,000$ distinct randomly generated weighted projective spaces satisfying the conditions in Assumption 1.4.1. In addition, the weighted projective spaces are required to have at worst terminal singularities. This check is done by using Propositions 1.1.4 and 1.1.3: the first result is only a necessary condition, but it helps to get rid of many examples efficiently. On the other hand, Proposition 1.1.3 is a necessary and sufficient condition, so it is used as the final check.

The examples are generated by choosing a random dimension $D \in \{1, \dots, 10\}$. Once the dimension is fixed, we generate a random example by choosing a random string of integers $[a_0, \dots, a_D]$ such that $a_0 \leq \cdots \leq a_D \leq 10D$. If the generated example does not satisfy Assumption 1.4.1 or it is not terminal, it is discarded and the process is repeated. The dataset is deduplicated by eliminating examples that are defined by the same string of integers (since we have assumed well-formedness – Condition 2 in Assumption 1.4.1 – this is enough to ensure we are not counting weighted projective spaces multiple times).

3.1.2 Toric varieties of Picard rank two

The dataset `DSdim_rk2` consists of $200\,000$ distinct randomly generated toric Fano varieties of Picard rank two satisfying the conditions in Assumption 1.4.1 and that have at worst terminal singularities (this check is performed in the naïve way described in Section 1.1.2).

In particular, the data generation process starts by choosing a dimension $D \in \{2, \dots, 10\}$ at random, and then generating a D-dimensional toric variety of Picard rank two as a $2 \times (D+2)$ weight matrix as in (1.1.2) with weights $0 \leq a_i, b_i \leq 5$. Unfortunately, deduplicating randomly-generated toric varieties of Picard rank two is a lot harder than deduplicating weighted projective spaces, because different weight matrices can give rise to the same toric variety. Therefore, we use the fan to deduplicate them according to the isomorphism type of Σ, by putting Σ in normal form; for details see [KS04, GK13].

Remark 3.1.1. When we generate Picard rank two data, we impose that each $2 \times N$ matrix contains an identity sub-block and all non-negative entries, i.e.

$$\begin{bmatrix} 1 & 0 & a_1 & \cdots & a_N \\ 0 & 1 & b_1 & \cdots & b_N \end{bmatrix}.$$

This implies that $C = \mathbb{R}^2_{\geq 0}$ in the formula for the regularized qunatum period (4.1.1), which simplifies the computational routine. This assumption is very mild, and all the theoretical results coming from the analysis on the dataset `DSdim_rk2` hold without assuming the presence of this identity sub-block.

3.2 Data analysis

In this section we discuss the preliminary data analysis performed on the regularized quantum period data of the toric Fano varieties in the datasets `DSdim_wps` and `DSdim_rk2`.

3.2.1 Weighted projective spaces

We compute an initial segment of the period sequence of all the terminal weighted projective spaces from the dataset `DSdim_wps`, $(c_0, \dots, c_m)$ where $m \approx 100\,000$, using (1.5.1). As a first step, we note that the non-zero coefficient of c_d appear to grow exponentially with d (we will give a rigorous proof of this statement in Chapter 4). Therefore, instead

DSdim_wps			DSdim_rk2		
Dimension	Sample size	Percentage	Dimension	Sample size	Percentage
1	1	0.001			
2	1	0.001	2	2	0.001
3	7	0.005	3	17	0.009
4	8 936	5.957	4	758	0.379
5	23 584	15.723	5	6 050	3.025
6	23 640	15.760	6	19 690	9.845
7	23 700	15.800	7	35 395	17.698
8	23 469	15.646	8	42 866	21.433
9	23 225	15.483	9	47 206	23.603
10	23 437	15.625	10	48 016	24.008
Total	150 000		Total	200 000	

Table 3.1: The number and percentage of terminal weighted projective spaces and toric varieties of Picard rank two appearing in datasets `DSdim_wps` and `DSdim_rk2`, by dimension.

of considering the coefficients of the period sequence, we consider $(\log c_d)_{d\in S}$ where $S = \{d \in \mathbb{Z}_{\geq 0} \mid c_d \neq 0\}$, which then appears to be growing linearly. As a dimensionality reduction step, we fit a linear model to the set $\{(d, \log c_d) \mid d \in S\}$ (see Section 2.3.1) and use the regression coefficients (slope and y-intercept) as the features in the machine learning models. In Figure 3.1 we depict a typical example of a terminal weighted projective space from `DSdim_wps` and highlight the linear behaviour of the logarithm of the non-zero terms of its period sequence.

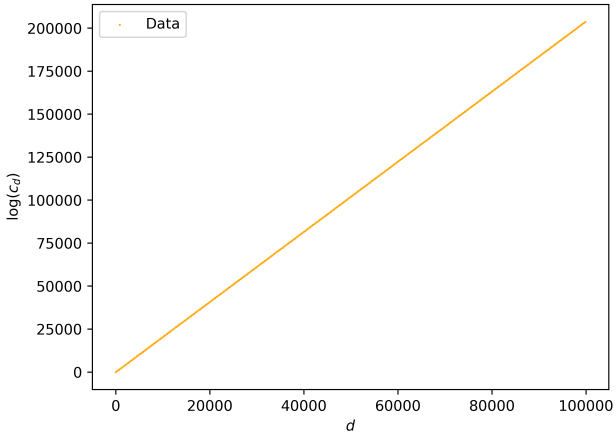


Figure 3.1: The logarithm of the non-zero period sequence coefficients c_d for a typical example: the weighted projective space $\mathbb{P}(5, 5, 11, 23, 28, 29, 33, 44, 66, 76)$.

After fitting a linear model to the logarithm of the non-zero coefficients of the period sequences, we can plot the slope against the y-intercept and colour datapoints by dimension to obtain Figure 3.2. The data clusters by dimension, and it is clearly linearly separable. This suggests that classification models such as Support Vector Machines should be able to determine the dimension from the slope and y-intercept alone.

To justify the use of linear regression we produce some error statistics demonstrating that the approximation is very accurate. The distribution of the standard errors for the slope

and the y-intercept are depicted in Figure 3.3 (recall the definitions (2.3.3) and (2.3.4)): the errors for the slope are between 3.9×10^{-8} and 1.4×10^{-5}, and the errors for the y-intercept are between 0.0022 and 0.82. This error increases as the dimension grows; see Figure 3.3c

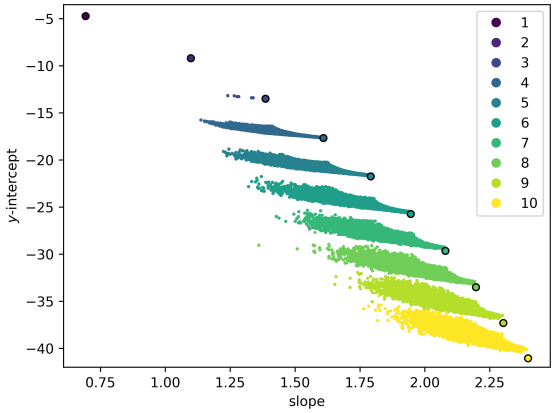
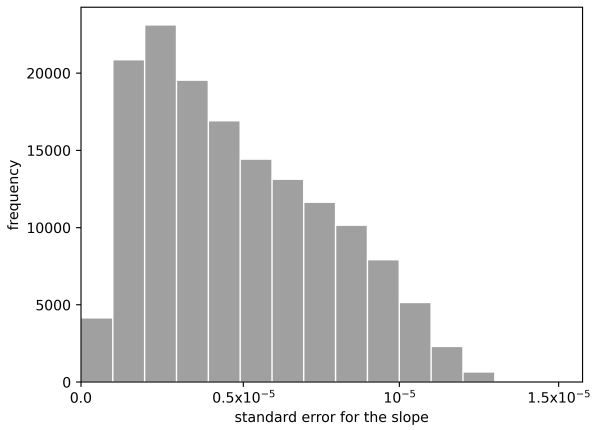
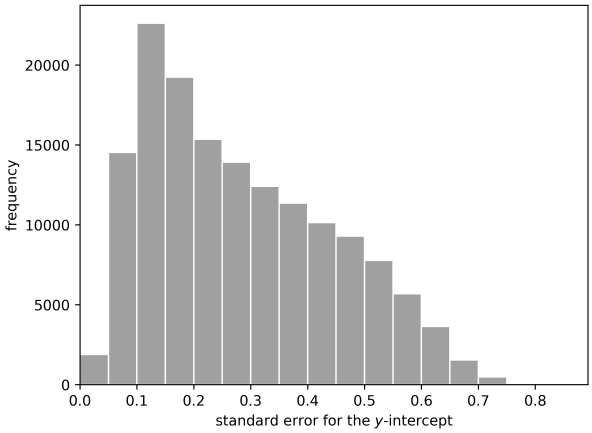


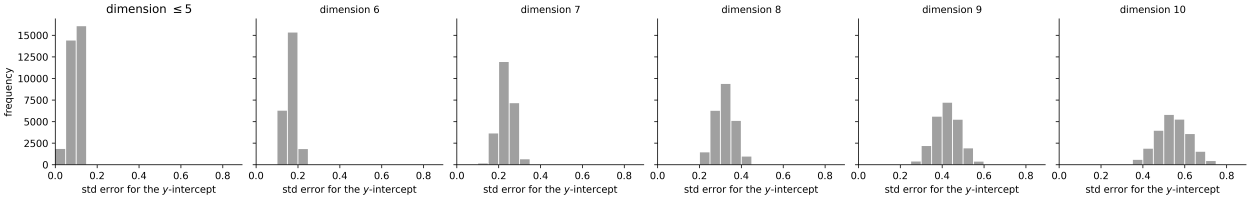
Figure 3.2: The slopes and y-intercepts from the linear model applied to the period sequences of terminal weighted projective spaces from `DSdim_wps`. The colour records the dimension of the weighted projective space. The circled bigger dots correspond to projective spaces $\mathbb{P}^n$ for $n = 1, \ldots, 10$.



(a) Standard error for the slope.



(b) Standard error for the y-intercept.

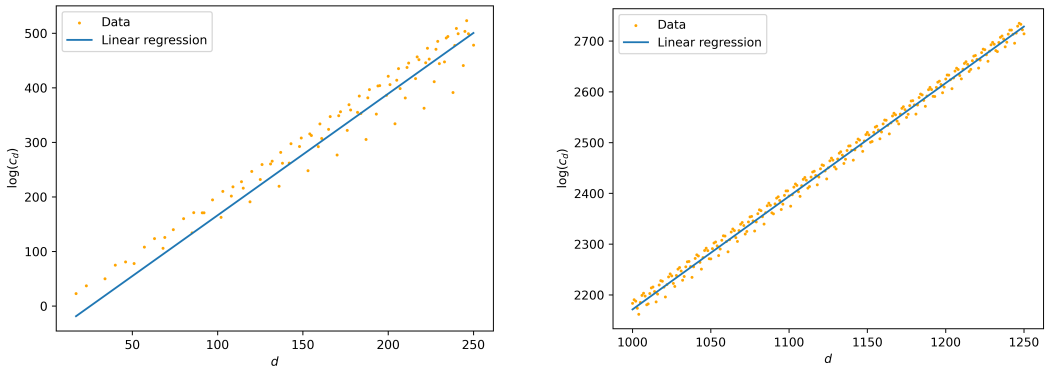


(c) Standard error for the y-intercept by dimension.

Figure 3.3: The distribution of standard errors for the slope and y-intercept from the linear model applied to the period sequences of terminal weighted projective spaces from `DSdim_wps`.

3.2.2 Toric varieties of Picard rank two

As for weighted projective spaces, we note that the period sequence $(c_d)_d$ of toric varieties of Picard rank two grows exponentially (again, we will prove this rigorously in Chapter 4). Therefore, we fit a linear model to $\{(d, \log c_d) \mid d \in S\}$ where $S = \{d \in \mathbb{Z}_{\geq 0} \mid c_d \neq 0\}$ and use the regression data (slope and y-intercept) as features in the machine learning models. See Example 3.2.1 for a typical example of growth of the logarithm of the non-zero terms of the period sequence.



(a) The first 250 terms.

(b) Terms between 1000 and 1250.

Figure 3.4: Different sections of the graph of the logarithm of the non-zero coefficients of the period sequence $(c_d)_d$ for Example 3.2.1, together with the line of best fit computed by the linear regression.

Example 3.2.1. Let us consider a typical example of a toric variety of Picard rank two, given by a weight matrix

$$\begin{bmatrix} 1 & 2 & 5 & 3 & 3 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 4 & 4 & 1 & 2 & 2 & 3 & 4 \end{bmatrix}.$$

In Figure 3.4 we plot the logarithm of the non-zero terms of its period sequence along with the linear approximation. Figure 3.4a shows only the first 250 terms, whilst Figure 3.4b shows the interval between the 1000th and the 1250th term. We see considerable deviation from the linear approximation amongst the first 250 terms; the deviation reduces for larger d.

For each example of toric variety of Picard rank two we compute its period sequence, using (1.5.2). Note that the formula is a lot more involved that the corresponding one for weighted projective spaces, hence in order to reduce computational costs, we compute pairs $(d, \log c_d)$ for $1000 \leq d \leq 20\,000$ by sampling every 100th term. Here, we discard the beginning of the period sequence because of the noise it introduces to the linear regression (as seen in Example 3.2.1). When sampling from the period sequence in this manner we might encounter some zero coefficients: when this occurs we instead consider the next non-zero coefficient.

After computing the logarithm of the period sequences in this manner, we extract the regression coefficients (slope and y-intercept) from the linear regression. Plotting slope against y-intercept and colouring datapoints by dimension we obtain Figure 3.2. Note that this figure is a lot less clear than Figure 3.2 – the equivalent for weighted projective spaces.

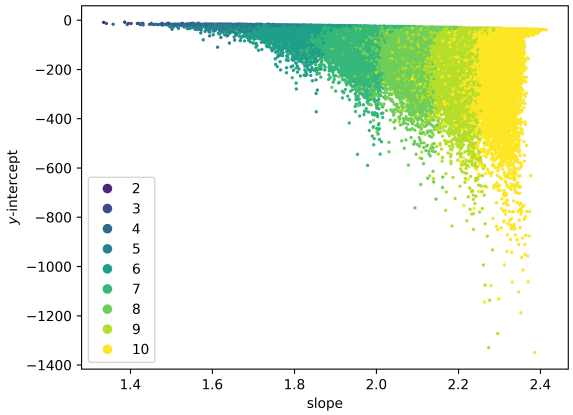
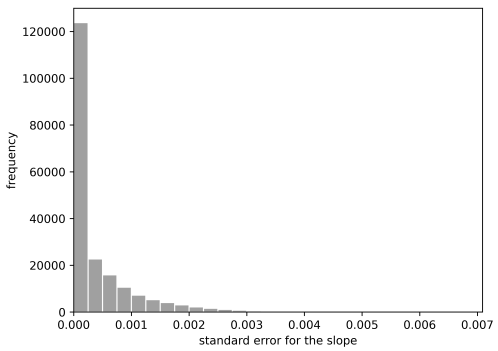


Figure 3.5: The slopes and y-intercepts from the linear model applied to the period sequences of toric Fano varieties from `DSdim_rk2`. The colour records the dimension of the toric variety.

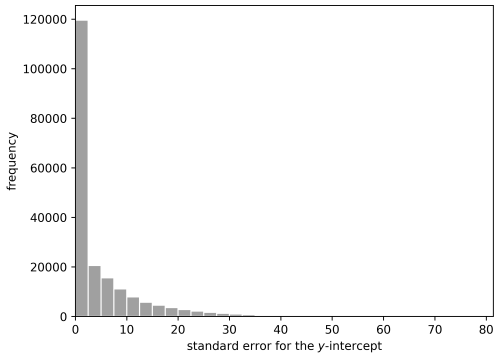
3.2.3 Restricting the standard error

Unlike for the dataset `DSdim_wps`, the study of the standard errors of the linear approximation for the dataset `DSdim_rk2` reveals that the errors present a larger variation. In Figure 3.6 we plot histograms for the standard errors of the slope and the y-intercept for `DSdim_rk2`. The standard errors for the slope are small compared to the range of slopes, but in many cases the standard error for the y-intercept is relatively large.

Discarding those samples that give rise to large standard errors for the y-intercept appears to reduce some noise, as shown in Figure 3.7. This observation appears to suggest that some underlying structure is obscured by the inaccuracies of the linear regressions: this is not surprising, since in the data generation step we might have undersampled from the period sequence.

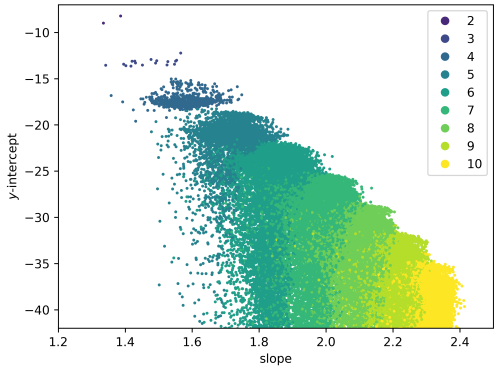


(a) Standard error for the slope.

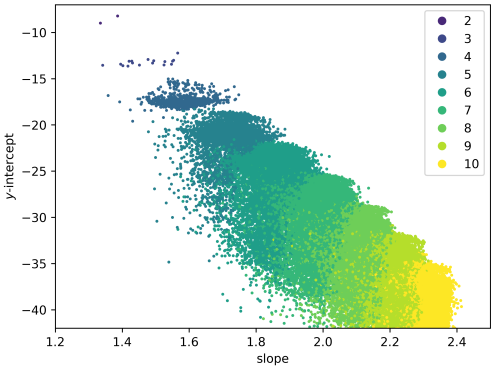


(b) Standard error for the y-intercept.

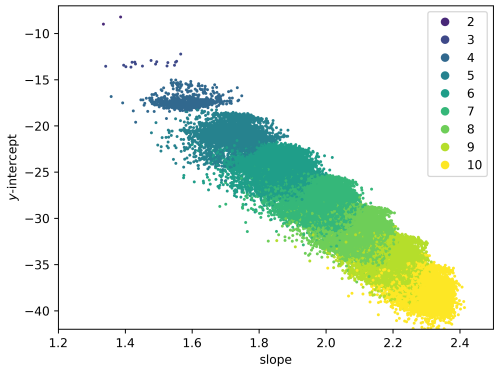
Figure 3.6: The distribution of standard errors for the slope and y-intercept from the linear model applied to toric varieties of Picard rank two with terminal singularities from `DSdim_rk2`.



(a) All data points.



(b) $s_{\text{int}} < 1$ (101 183/200 000 points).



(c) $s_{\text{int}} < 0.3$ (67 445/200 000 points).

Figure 3.7: The slopes and y-intercepts from the linear model applied to toric varieties of Picard rank two from `DSdim_rk2`, selecting data points according to the standard error s_{int} for the y-intercept. The colour records the dimension of the toric variety.

Example 3.2.2. Consider the five-dimensional toric variety with Picard rank two and weight matrix

$$\begin{bmatrix} 1 & 10 & 5 & 13 & 8 & 12 & 0 \\ 0 & 0 & 3 & 8 & 5 & 14 & 1 \end{bmatrix}. \tag{3.2.1}$$

This example is one of the outliers in Figure 3.5. The regression coefficients obtained by fitting a linear model to the logarithm of the non-zero coefficients of the period sequence are 1.637, for the slope, and -62.64, for y-intercept. In order to test how the behaviour of the standard error of the y-intercept changes, we compute the first $40\,000$ coefficients c_d as in (1.5.2) (without sampling every 100 terms). As shown in Figure 3.9b, as d increases the y-intercept of the linear model increases to -28.96 and the corresponding standard error (s_{int}) decreases to 0.7877. On the other hand, the slope and corresponding error remains more or less unchanged. This is positive evidence for the theory that bad linear fit are a result of our data generation process, where we might be undersampling the sequence and not computing enough terms. We expect that generating (many) more coefficients of the period sequence would significantly reduce noise in Figure 3.5, but it remains impractical computationally – see Figure (3.2.1) for a comparison of the timing data for computing different numbers of terms in the period sequence ($40\,000$ terms takes roughly 100 times longer than $10\,000$ terms). Moreover, note that in this example even $40\,000$ coefficients may not be enough (for example it still gets misclassified by the Support Vector Machine as a

six-dimensional example, see Figure 3.23 and later discussion).

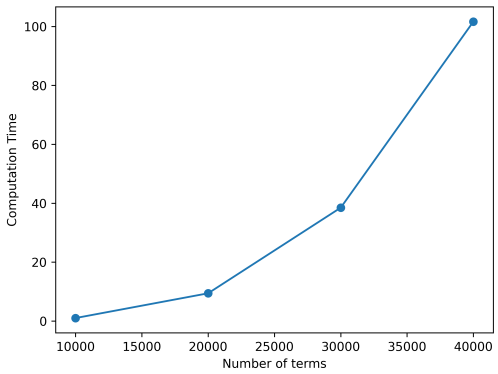
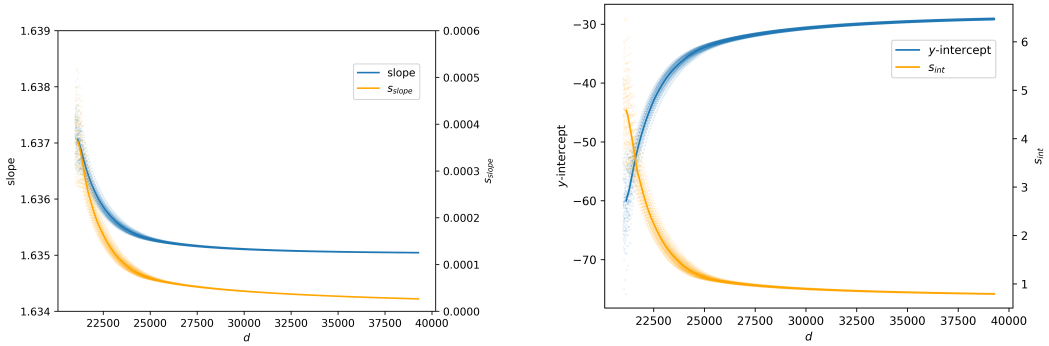


Figure 3.8: Timings for computing $10\,000, 20\,000, 30\,000, 40\,000$ terms of the period sequence of the weight matrix (3.2.1). The timings are scaled so $t_{10\,000} = 1$.



(a) For the slope

(b) For the y-intercept.

Figure 3.9: The slope and the y-intercept, and their respective standard errors for Example 3.2.2, computed from pairs $(k, \log c_k)$ such that $d - 20\,000 \leq k \leq d$ by sampling every 100th term, together with a LOWESS-smoothed trend line.

We remark that computing more coefficients of the period sequence for the whole dataset might result in improved standard errors for the y-intercept. However, this approach is impractical from the point of view of computation time, therefore we restrict to those toric varieties of Picard rank two such that the y-intercept standard error s_{int} is less than 0.3; see Table 3.2 for the data breakdown by dimension.

3.3 Machine learning

In this section, we build machine learning classifiers that predict the dimension of varieties in datasets `DSdim_wps` and `DSdim_rk2` from data coming from their period sequences. The experiments were conducted using scikit-learn [PVG+11], a standard machine learning library for Python. For all models, we fix the most promising hyperparameters by trying many configurations using *randomised grid search*. The code used to perform this analysis is available from Bitbucket [CKV22c] under an MIT licence.

DSdim_rk2 with $s_{\text{int}} < 0.3$		
Dimension	Sample size	Percentage
3	17	0.025
4	758	1.124
5	5 504	8.161
6	12 497	18.530
7	16 084	23.848
8	13 701	20.315
9	10 638	15.773
10	8 244	12.224
Total	67 443	

Table 3.2: The number and percentage of toric varieties of Picard rank two with $s_{\text{int}} < 0.3$ appearing in DSdim_rk2, by dimension.

3.3.1 Weighted projective spaces

We consider the dataset DSdim_wps excluding dimensions one and two (since there is only one terminal weighted projective space in each class, namely $\mathbb{P}^1$ and $\mathbb{P}^2$), so the data is reduced to $149\,998$ samples. We do not work directly with the regularized quantum periods, but instead we set the features to the regression data (pairs of slope and y-intercept) labelled by dimension, which varies between three and ten.

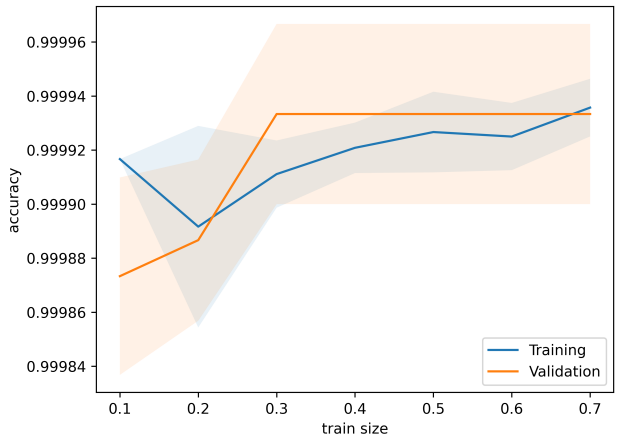


Figure 3.10: Learning curves for a linear SVM applied to data coming from DSdim_wps (excluding dimensions one and two). The plot shows the means of the training and validation accuracies for five different random train–test splits. The shaded regions correspond to the 1σ interval, where σ denotes the standard deviation.

We apply a Support Vector Machine (SVM) with linear kernel and regularisation parameter $C = 10$ on the standardised features (obtained by translating the mean to zero and scaling the variance to one). Figure 3.10 shows the learning curves for different train–test splits: here the confidence intervals are given by the standard deviation calculated for five random train–test split each time. Using 10% of the data for training we obtained an accuracy of 99.99%. This very accurate SVM gives us linear decision boundaries that separate the different clusters of examples of different dimension, as shown in Figure 3.11.

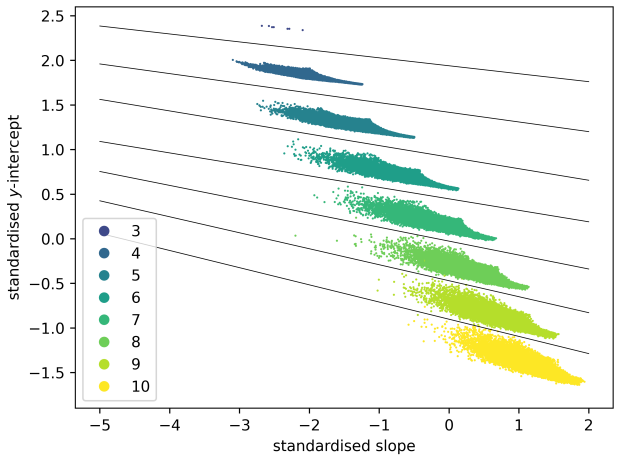


Figure 3.11: Decision boundaries computed from a linear SVM trained on 70% of the data coming from `DSdim_wps` (excluding dimensions one and two). Note that the data has been standardised.

3.3.2 Toric varieties of Picard rank two

As discussed in Section 3.2.3, we restrict attention to toric varieties with Picard rank two such that the y-intercept standard error (s_{int}) is less than 0.3. We also exclude dimension two from the analysis, since in this case there are only two varieties (namely, $\mathbb{P}^1 \times \mathbb{P}^1$ and the Hirzebruch surface $\mathbb{F}_1$). The resulting dataset contains $67\,443$ pairs of slope and y-intercept, labelled by dimension; the dimension varies between three and ten, as shown in Table 3.2.

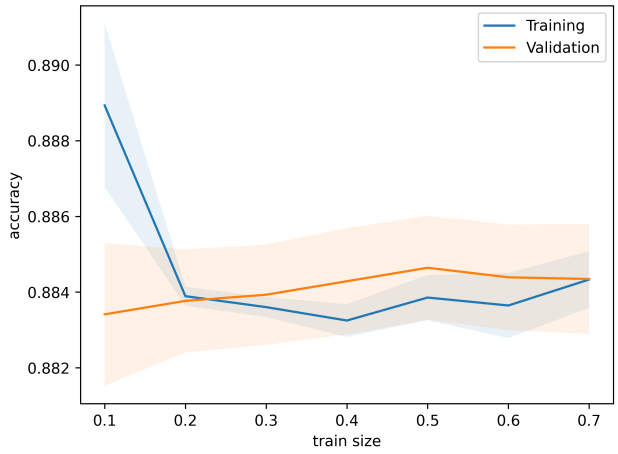


Figure 3.12: Learning curves for a linear SVM applied to the data coming from `DSdim_rk2` (with $s_{\text{int}} < 0.3$ and excluding dimension two). The plot shows the means of the training and validation accuracies for five different random train–test splits. The shaded regions correspond to the 1σ interval, where σ denotes the standard deviation.

Support Vector Machine We use an SVM with linear kernel and regularisation parameter $C = 50$ on `DSdim_rk2` (with $s_{\text{int}} < 0.3$ and excluding dimension two). By considering different train–test splits we obtained the learning curves shown in Figure 3.12, where the means and the standard deviations were obtained by performing five random samples for

each split. Note that the model does not overfit. We obtain a validation accuracy of 88.20% using 70% of the data for training. Figure 3.13 shows the decision boundaries computed by the SVM between neighbouring dimension classes. Figure 3.14 shows the confusion matrices for the same train–test split.

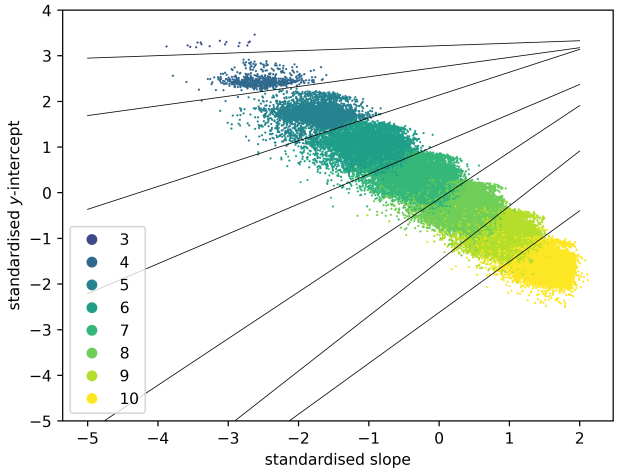


Figure 3.13: Decision boundaries computed from a linear SVM trained on 70% of the data coming from `DSdim_rk2` (with $s_{\text{int}} < 0.3$ and excluding dimension two). Note that the data has been standardised.

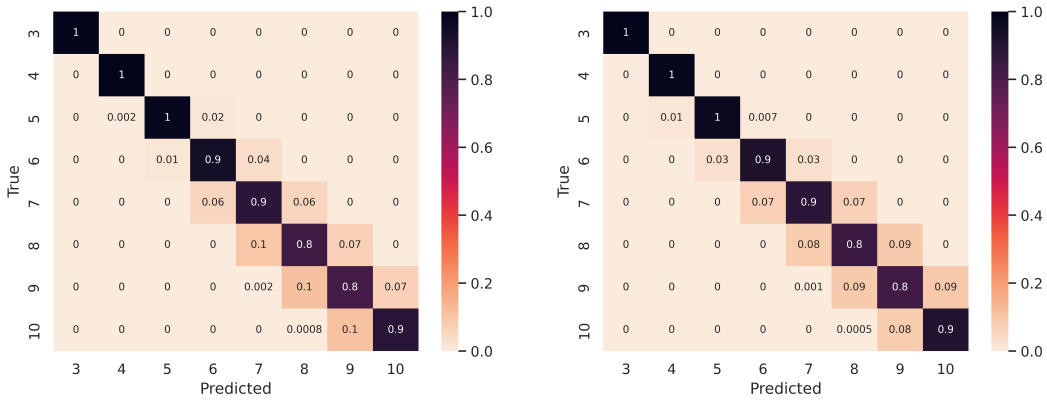
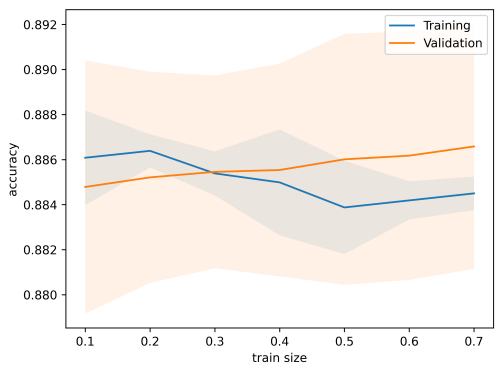


Figure 3.14: Confusion matrices for a linear SVM trained on 70% of the data coming from `DSdim_rk2` (with $s_{\text{int}} < 0.3$ and excluding dimension two): (a) is normalised with respect to the true axis; (b) is normalised with respect to the predicted axis.

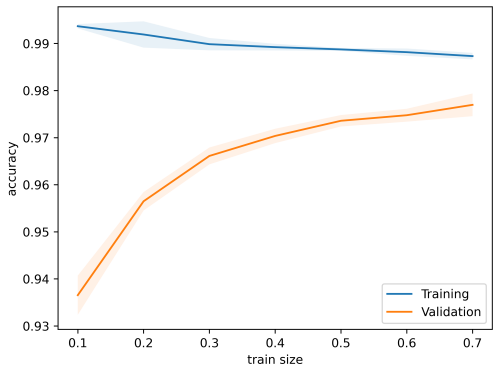
Neural Networks Neural networks do not handle unbalanced datasets well. Therefore, we remove those terminal weighted projective spaces with dimensions two, three, four, and five from our `DSdim_rk2` (with $s_{\text{int}} < 0.3$) and call this dataset `DSdim_rk2_cut`. We train a Multilayer Perceptron (MLP) (recall the architecture from Section 2.3.3) classifier on the same features (slope and y-intercept), using an MLP with three hidden layers $(10, 30, 10)$, Adam optimiser [KB14], and ReLU activation function (see Example 2.3.1). The learning curve in Figure 3.15a is produced by using different train–test splits, and again note that

the model does not overfit. With 70% of the data for training the MLP gave a validation accuracy of 88.7%. We refer to this network as MLP_2.

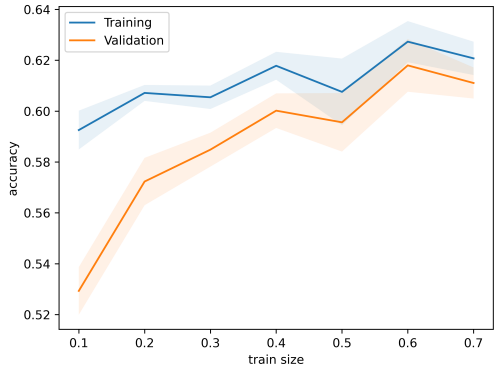
Moreover, in order to work with more balanced data, we have performed random undersampling on the data so that there are the same number of samples in each dimension (8244 representatives). We do so using a Python undersampling library imalanced-learn [LNA17]. This however does not improve the accuracy of the model: the benefit brought by a better balanced dataset was outweighed by the loss of data caused by the undersampling. Therefore, the results we will discuss are not performed on a perfectly balanced dataset.



(a) MLP_2.



(b) MLP_{102}.



(c) MLP_{100}.

Figure 3.15: Learning curves for the classifiers MLP_2, MLP_{102}, MLP_{100} applied to data coming from `DSdim_rk2_cut`. The plot shows the means of the training and validation accuracies for five different random train–test splits. The shaded regions show the 1σ interval, where σ is the standard deviation.

In order attempt to get a better performing model, we also train an MLP with the same architecture but supplementing the features by including some period sequence coefficients. The features are the two regression coefficients and $\log c_d$ for $1 \leq d \leq 100$ (setting this quantity to zero when $c_d = 0$). We refer to this neural network as MLP_{102} since it has 102 features. The learning curve for different train–test splits is in Figure 3.15b. Using 70% of the data for training, the MLP_{102} model gave a validation accuracy of 97.7%.

It is unclear the reason why MLP_{102} performs better than MLP_2. One possible explanation, and the underlying reason why we have included the first 100 terms of the period sequence rather than other portions of it, is that the beginning of the period sequence exhibits a more

noisy oscillatory behaviour. This behaviour decays as d grows – see Figures 3.4. This initial part reduces the accuracy of the linear regression (which is the reason why we had originally excluded it), but it might still carry important information. For example, by examining for which index c_d is zero at the beginning of the period sequence, it is sometimes possible to recover the values of a and b (the sum of the top and bottom row in the weight matrix as in (1.1.2)) – see Section 6.2 for more details. This type of information is helpful: for example larger values of a and b usually go along with a larger dimension. Note however, that most of the information is encoded in the slope and y-intercept. This is highlighted by the SHAP plot in Figure 3.16, which analyses the model sensitivity to the input features. This analysis suggests that the dimension of X should be visible from a rigorous estimate of the asymptotic behaviour of $\log(c_d)$.

Moreover, training an MLP on just the coefficients $\log c_d$ for $1 \leq d \leq 100$ with the same architecture (we refer to this MLP as MLP_{100}) gives an accuracy of only 62%, indicating even more strongly that the regression data plays a fundamental role in determining the dimension; the learning curves are depicted in Figure 3.15c.

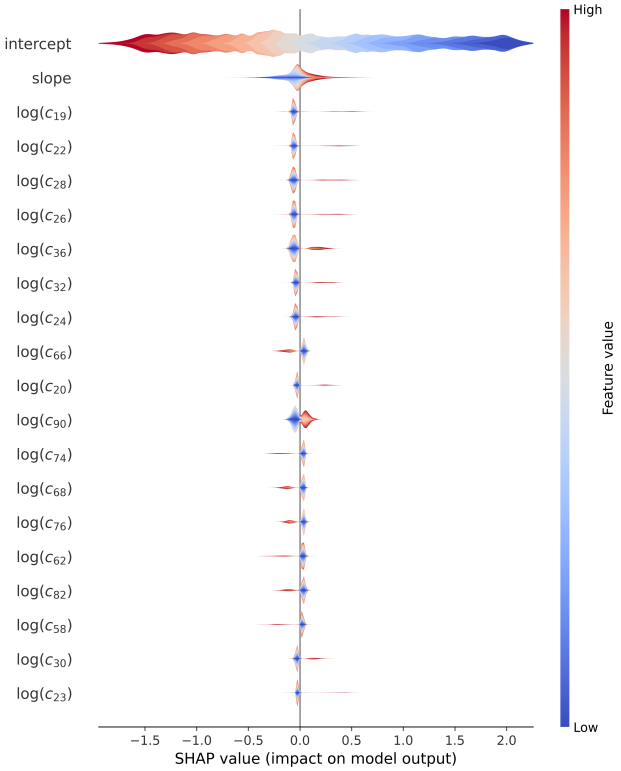


Figure 3.16: Model sensitivity analysis using SHAP values, for the model MLP_{102} trained on the data coming from `DSdim_rk2_cut`. It predicts the dimension with 97.7% accuracy.

Comparison of models We compare the test accuracies of all the different models (the SVM and the three neural networks MLP_2, MLP_{100}, and MLP_{102}) in Table 3.3, against baseline (i.e. always predicting the most populated class – dimension seven). These accuracies are obtained by training and testing on 70% of the same dataset, the data coming from `DSdim_rk2_cut`. Moreover, we note that misclassified examples across all models are usually in high dimension, which is consistent with the theory that the errors in the classifi-

cation are related to the data generation workflow making it harder to read off the asymptotic behaviour from the linear regression. In fact, higher dimension examples tend to have higher a and b. Since the non-zero elements of the period sequence depend on integer linear combinations of a and b, we note that there is a correlation between high-dimensional examples and the need for more terms of the period sequence in order to get to a more regular behaviour.

We compare their confusion matrices in Table 3.4.

ML models				
Baseline	SVM	MLP_2	MLP_{100}	MLP_{102}
23.8%	87.7%	88.7%	62.0%	97.7%

Table 3.3: Comparison of model accuracies. Accuracies for various models applied to the data coming from `DSdim_rk2_cut`: a linear SVM and the neural networks MLP_2, MLP_{100}, and MLP_{102}. They are compared with the baseline accuracy of always predicting the most populated class (dimension seven).

3.4 Asymptotic behaviour

The above analysis strongly suggests that the asymptotic behaviour of $\log(c_d)$ should contain information about the dimension. It inspires the conjecture of rigorous asymptotics for the regularized quantum period of weighted projective spaces and Picard rank two toric Fano varieties which makes this dependence clear. We state here the results as in [CKV23b] (where they are also proven), but we prove the more general version of these statements in Chapter 4 for all toric Fano varieties (Theorem 4.1.2).

Theorem 3.4.1 (Theorem 5, [CKV23b]). *Let X be the weighted projective space $\mathbb{P}(a_0, \ldots, a_N)$ satisfying Assumption 1.4.1, so that the dimension of X is N. Let c_d denote the coefficient of t^d in the regularized quantum period $\widehat{G}_X(t)$ (as in (1.5.1)). Let $a = a_0 + \cdots + a_N$. Then $c_d = 0$ unless d is divisible by a, and*

$$\log c_d \sim Ad - \frac{\dim X}{2} \log d + B$$

as $d \to \infty$, where

$$A = -\sum_{i=0}^{N} p_i \log p_i \qquad B = -\frac{\dim X}{2} \log(2\pi) - \frac{1}{2} \sum_{i=0}^{N} \log p_i \tag{3.4.1}$$

and $p_i = a_i/a$.

Theorem 3.4.2 (Theorem 6, [CKV23b]). *Let X be a toric variety of Picard rank two and dimension $N-2$ given as a weight matrix*

$$\begin{bmatrix} a_1 & a_2 & a_3 & \cdots & a_N \\ b_1 & b_2 & b_3 & \cdots & b_N \end{bmatrix}$$

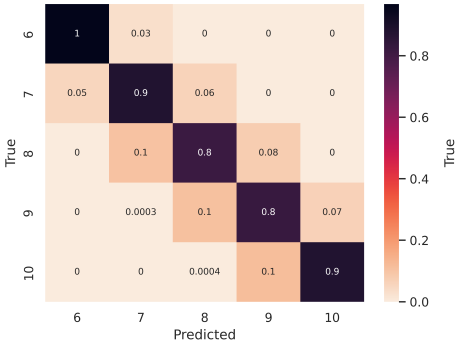
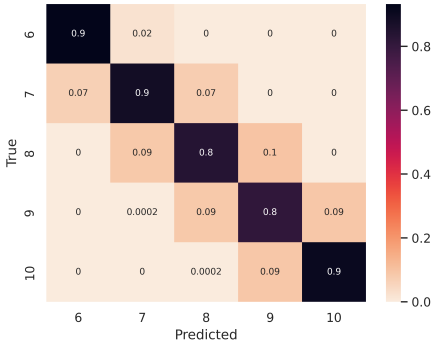
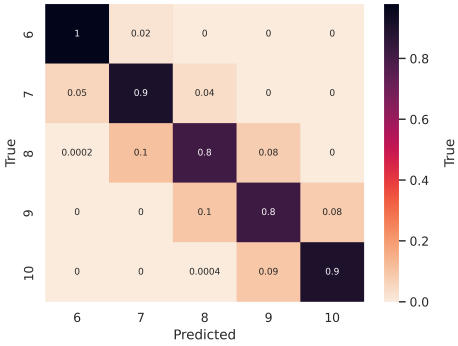
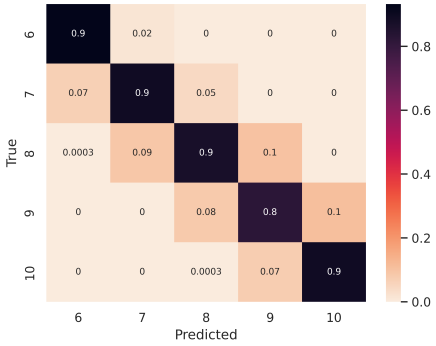
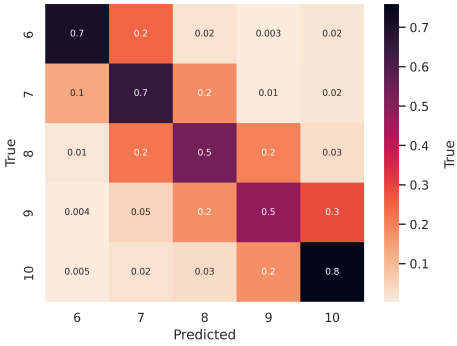
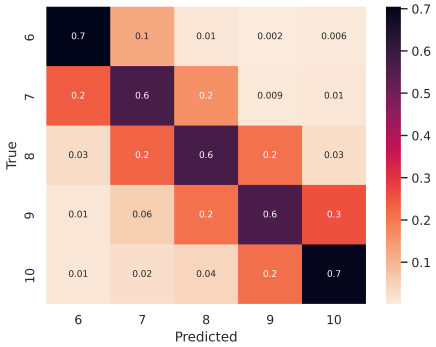
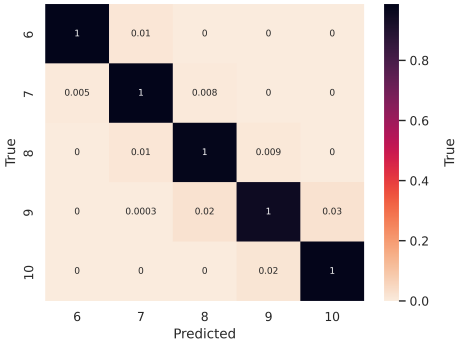
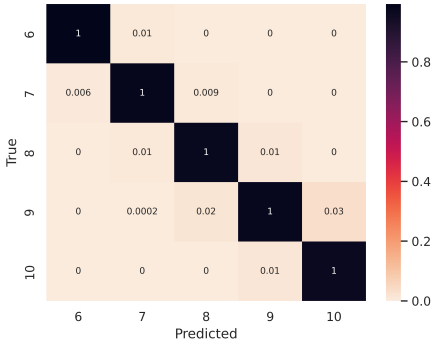
Model	True confusion matrix	Predicted confusion matrix
SVM		
MLP_2		
MLP_{100}		
MLP_{102}		

Table 3.4: Comparison of confusion matrices. Confusion matrices for various models applied to the data coming from `DSdim_rk2`: a linear SVM and the neural networks MLP_2, MLP_{100}, and MLP_{102}. The first column is normalised with respect to the true axis; the second column is normalised with respect to the predicted axis.

satisfying Assumption 1.4.2. Let $a = a_1 + \cdots + a_N$ *and* $b = b_1 + \cdots + b_N$*, and let* $[\mu : \nu] \in \mathbb{P}^1$ *be the unique solution to the bivariate homogeneous equation*

$$\begin{gathered}\prod_{i=1}^{N}(a_i\mu + b_i\nu)^{a_i b} = \prod_{i=1}^{N}(a_i\mu + b_i\nu)^{b_i a} \\ \binom{\mu}{\nu} \in C\end{gathered} \tag{3.4.2}$$

where

$$C = \{x \in \mathbb{R}^r \mid \alpha_i^T \cdot x \geq 0 \text{ for all } i = 1, \ldots, N\}\,.$$

Let c_d *denote the coefficient of* t^d *in the regularized quantum period* $\widehat{G}_X(t)$ *(as in* (1.5.2)*). Then non-zero coefficients* c_d *satisfy*

$$\log c_d \sim Ad - \frac{\dim X}{2}\log d + B$$

as $d \to \infty$*, where*

$$\begin{aligned} A &= -\sum_{i=1}^{N} p_i \log p_i \\ B &= -\frac{\dim X}{2}\log(2\pi) - \frac{1}{2}\sum_{i=1}^{N}\log p_i - \frac{1}{2}\log\left(\sum_{i=1}^{N}\frac{(a_i b - b_i a)^2}{\ell^2 p_i}\right) \end{aligned} \tag{3.4.3}$$

and $p_i = \dfrac{\mu a_i + \nu b_i}{\mu a + \nu b}$.

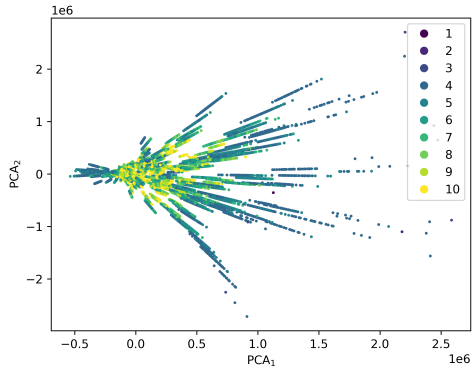
Note that neither these results assume terminality, but they hold as long as our varieties are $\mathbb{Q}$-factorial.

3.5 Experiments with principal component analysis

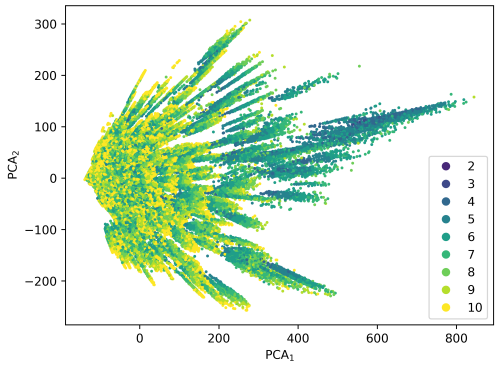
So far in this chapter we have relied on applying linear regression to reduce the dimensionality of the data coming from the regularized quantum period of the algebraic varieties of interest. However, a standard approach to dimensionality reduction is Principal Component Analysis (PCA); see Section 2.2.2. In this section we explore a PCA approach to dimensionality reduction and motivate our choice to not consider it in our machine learning analysis.

We perform four experiments using PCA. We consider only the first two components of the PCA $(\text{PCA}_1, \text{PCA}_2)$ in an attempt to reproduce plots resembling the clustering behaviour from Figures 3.2 and 3.5, which were obtained by looking at the linear regression output.

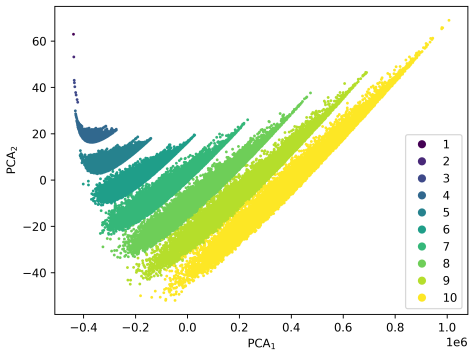
1. The first experiment fits a two component PCA on the first $100\,000$ terms of the sequence $(c_d)_d$ originating from `DSdim_wps`, taking the logarithm of the non-zero terms



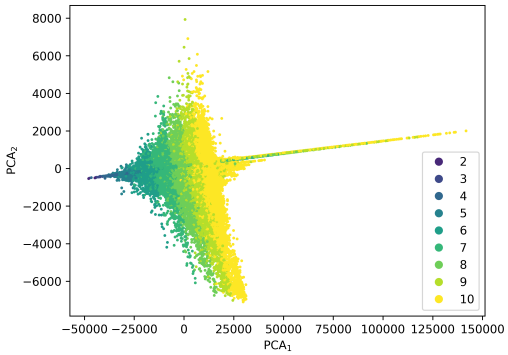
(a) PCA components for `DSdim_wps` including zeros (experiment 1).



(b) PCA components for `DSdim_wps` excluding zeros (experiment 2).



(c) PCA components for `DSdim_rk2` including zeros (experiment 3).



(d) PCA components for `DSdim_rk2` excluding zeros (experiment 4).

Figure 3.17: Plot of the two PCA components of the regularized quantum period data from `DSdim_wps` and `DSdim_rk2`, either including or excluding those values that are zeros. Each data point is coloured by dimension.

and leaving the zero terms unchanged. This approach is not very successful, and was very computationally expensive (as we have to perform partial fits for the PCA because the data does not fit in memory). The PCA explained variance ratios for the first two components are $(0.014, 0.011)$, and when we plot PCA_1 against PCA_2 in Figure 3.17a we note that the clusters are not linearly separable.

2. The second experiment fits a two-components PCA to the first 100 terms of the sequence $(c_d)_d$ for dataset `DSdim_rk2`, taking the logarithm of the non-zero terms and leaving the zero terms unchanged. This approach is again not very successful: the PCA explained variance ratios are $(0.148, 0.053)$, and when we plot PCA_1 against PCA_2 in Figure 3.17b we note that there are no clear clusters.

3. The third experiment uses $(\log c_d)_d$ coming from `DSdim_wps`, where in each case d ranges over the sequence such that $c_d \neq 0$. Depending on the weights, the indices corresponding to non-zero c_d vary – c_d is non-zero only if d is divisible by the Fano index r of the corresponding weighted projective space. So in order to fit the PCA we consider only the first m non-zero terms of each weighted projective space, where m is the minimal number of non-zero terms encountered overall in `DSdim_wps`. Without knowing the statement of Theorem 3.4.1 it is unclear why this would be a sensible approach, since *a priori* we are comparing terms c_d for different values of d. This experiment is successful, with PCA explained variance ratios $(0.999, 0.216 \times 10^{-9})$ and the clusters in Figure 3.17c, obtained when plotting PCA_1 against PCA_2, revealing a 'stretched out' version of Figure 3.2. The clusters are clearly linearly separable.

4. The fourth experiment fits a PCA with two components to $(\log c_d)_d$ coming from `DSdim_rk2`. Again, depending on the weights, the indices corresponding to non-zero c_d varies, so in order to fit the PCA we consider only the first m non-zero terms of each, where m is the minimal number of non-zero terms encountered. Again, without knowing the statement of Theorem 3.4.2 it is unclear why this would be meaningful, for the same reason as above. This experiment was more successful than experiment 2, with PCA explained variance ratios $(0.993, 0.004)$ and the clusters in Figure 3.17d, obtained when plotting PCA_1 against PCA_2, somewhat reminiscent of Figure 3.5.

Note that we can improve experiment 3 by considering the different patterns of zeros in the period sequence, depending on the Fano index r. Therefore, the correct way of using PCA would be to perform PCA for Fano varieties of each Fano index r separately. When doing this, we note that for each r the first two components of PCA are related to the growth coefficients (A, B) from Theorem 3.4.1 by an invertible affine-linear transformation. In Table 3.5 we record some of the affine-linear transformation coefficients $\alpha_r, \beta_r, \gamma_r$ that describe the following relationships

$$\begin{aligned} \text{PCA}_1 &\sim \alpha_r^1 A + \beta_r^1 B + \gamma_r^1 \,, \\ \text{PCA}_2 &\sim \alpha_r^2 A + \beta_r^2 B + \gamma_r^2 \,. \end{aligned}$$

Therefore, by performing this analysis, we conclude that the first two components of the PCA seem to carry an equivalent amount of information as the coefficients A and B. However, using A and B as features has major benefits compared to using the PCA coefficients. First, they allow us to make a meaningful comparison between Fano varieties of different

	PCA$_1$				PCA$_2$		
r	α_r^1	β_r^1	γ_r^1	r	α_r^2	β_r^2	γ_r^2
1	1.23×10^6	18.3	-2.21×10^6	1	-2.22×20^2	-10.9	1.18×10^2
2	1.14×10^6	17.2	-2.13×10^6	2	-2.20×20^2	-10.1	1.33×10^2
3	1.20×10^6	17.9	-2.18×10^6	3	-2.18×20^2	-10.6	1.21×10^2
4	1.18×10^6	17.7	-2.16×10^6	4	-1.93×20^2	-10.4	0.78×10^2
5	1.31×10^6	19.7	-2.29×10^6	5	-2.21×20^2	-10.6	1.05×10^2

Table 3.5: Affine transformations between (A, B) and the first two principal components $(\text{PCA}_1, \text{PCA}_2)$ obtained by performing PCA on $(c_d)_d$ for Fano indices between $r = 1, 2, 3, 4, 5$.

Fano index. Secondly, unlike PCA-derived values, they can be computed for a single Fano varieties, while the PCA needs to be fitted on a large number of Fano varieties of the same index before it can be used. We also expect a similar analysis can be performed on the Picard rank two data.

3.6 Theoretical analysis

Having rigorous asymptotic formulae from Theorems 3.4.1 allows us to revisit the clustering behaviour from Figure 3.2. Instead of using the data coming from the linear regression (i.e. the slope and the y-intercept), we use the coefficients A and B appearing in the asymptotic expression

$$\log c_d \sim Ad - \frac{\dim X}{2} \log d + B \, .$$

For weighted projective spaces, the formulae for A and B are specified in Theorem 3.4.1. Figure 3.18 shows the values of A and B for all terminal weighted projective spaces (with weights bounded by 25) and dimension between one and ten, coloured by dimension. Note the clusters which now overlap.

As an example, let us consider the cluster for weighted projective spaces of dimension five, which are reproduced in Figure 3.19. We can produce linear upper and lower bounds in the following way. For a suitable $\theta \geq 0$ we can consider

$$B + \theta A = -\frac{\dim X}{2} \log(2\pi) - \frac{1}{2} \sum_{i=1}^{N} \log p_i - \theta \sum_{i=1}^{N} p_i \log p_i$$

where $\dim X = N = 5$. The lower bound is obtained by solving the constrained optimisation problem

$$\min(B + \theta A) \qquad \text{subject to} \qquad \begin{array}{l} p_0 + \cdots + p_5 = 1 \\ p_0, \ldots, p_5 \geq 0 \end{array}$$

on the five-simplex. Note that we are not assuming terminality to construct this bound.

However, we are unable to construct an upper bound in a similar way, since B is unbounded, meaning that $B + \theta A$ must also be. However, we can consider the following

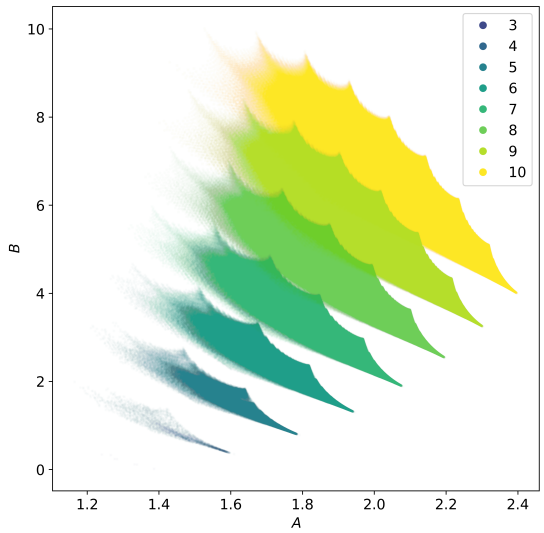


Figure 3.18: The values of A and B for all weighted projective spaces $\mathbb{P}(a_0, \ldots, a_N)$ with terminal singularities and $a_i \leq 25$ for all i, coloured by dimension.

constrained optimisation problem for a small positive $\epsilon > 0$

$$\max(B + \theta A) \qquad \text{subject to} \qquad \begin{array}{l} p_0 + \cdots + p_5 = 1 \\ \epsilon \leq p_0 \leq \cdots \leq p_5 \, . \end{array}$$

Note that ϵ can be taken as $1/a$ (for a the maximum sum of the weights considered). In our case $a = 124$, since all our weights are bounded by 25 (and they cannot all be 25 because of the well-formedness assumption, Condition 2 in Assumption 1.4.1). This value is the one taken in Figure 3.19. We know that such an a must exist since there are finitely many terminal weighted projective spaces (by boundedness of Fanos with bounded complexity on the singularities [Bir21]). Having this value will give a linear upper bound for the cluster. This is summarised in the following Proposition.

Proposition 3.6.1 (Illustrated in Figure 3.19)**.** *Let X be the five-dimensional weighted projective space $\mathbb{P}(a_0, \ldots, a_5)$, and let A, B be as in Theorem 3.4.1. Then $B + \frac{5}{2}A \geq \frac{41}{8}$. If in addition $a_i \leq 25$ for all i then $B + 5A \leq \frac{41}{40}$.*

Similarly, we could obtain linear bounds for the clusters in each dimension, however, as Figure 3.18 shows the clusters are not linearly separable. This is in contrast with Figure 3.2 where the clusters were separated by the linear boundaries from the linear SVM. This can be justified in the following way. The actual asymptotics of the regularized quantum period do not take into account the term $-\frac{D}{2}\log(d)$, where D is the dimension of the variety. Note that this term does not vary too much in the range of degrees considered, i.e. $d < 100\,000$. Therefore, we could consider the y-intercept resulting from the linear regression as being approximated by $B - \frac{11}{2}\dim X$ ($\log(100\,000) \sim 11$). This extra term distorts the clusters and translates them, making them linearly separable. A pictorial representation of this is given by Figure 3.21.

We expect that the same mechanism applies in Picard rank two as well; see Figure 3.20.

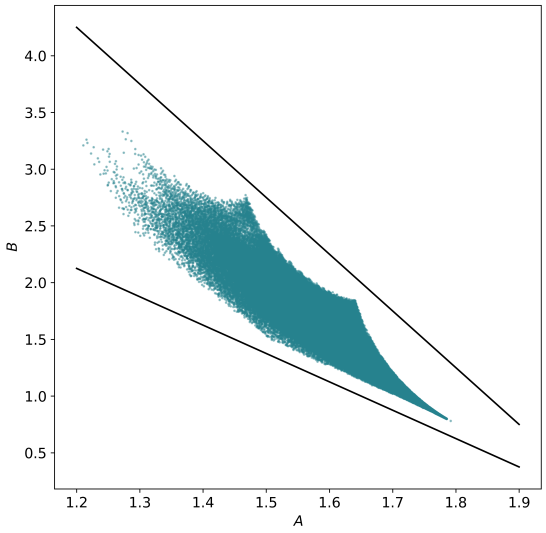


Figure 3.19: Linear bounds for the cluster given by weighted projective spaces $\mathbb{P}(a_0, \ldots, a_5)$ with terminal singularities and $a_i \leq 25$ for all i, given by Proposition 3.6.1.

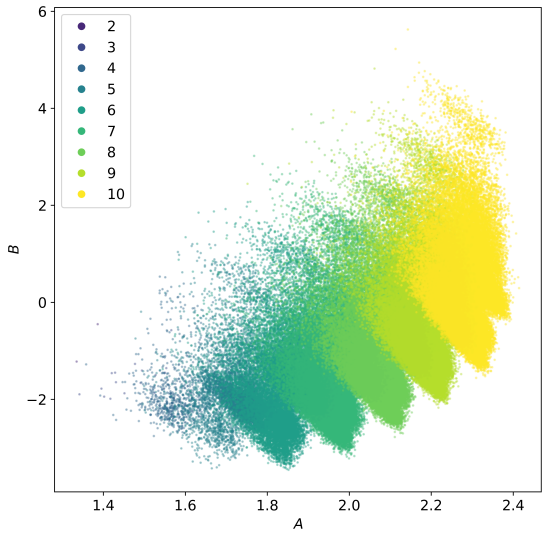


Figure 3.20: The values of A and B for toric varieties of Picard rank two in the dataset `DSdim_rk2`, coloured by dimension.

However, we note that generating large numbers of Picard rank two terminal toric Fano varieties is hard, because checking terminality in higher rank relies on lattice point counts (see Section 1.1.2 for details). So, we are currently unable to produce a more comprehensive picture for Picard rank two variety, as the one for weighted projective spaces. We explore how to address this problem further in Chapter 5.

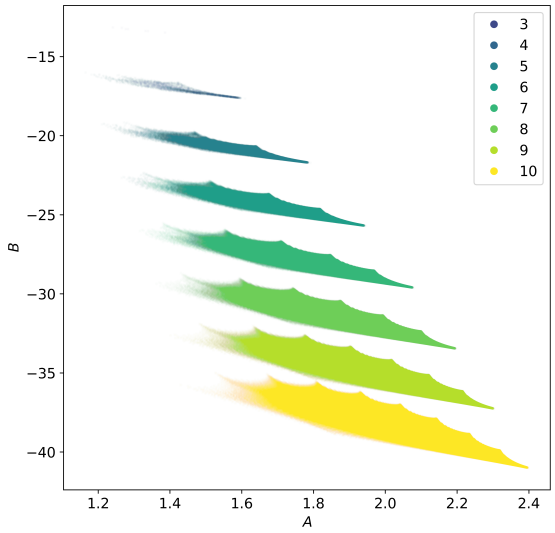


Figure 3.21: The values of A and $B - \frac{11}{2}\dim X$ for all weighted projective space $\mathbb{P}(a_0, \ldots, a_N)$ with terminal singularities and $a_i \leq 25$ for all i, coloured by dimension.

3.7 Outlook

In this chapter we have built models that predict the dimension of weighted projective spaces and Picard rank two toric Fano varieties using data coming from their regularized quantum periods. This is proof of concept and evidence for a fundamental conjecture in the Fano classification program: that the regularized quantum period of a Fano variety determines that variety. The machine learning experiments guided the construction of rigorous asymptotic formulae that make clear the dependency of the period sequence on the dimension, in the case we have studied. However, somewhat perversely, because the series involved converge extremely slowly, reading the dimension of a Fano variety directly from the asymptotics of the regularized quantum period is not practical.

For instance, consider the Fano variety from Example 3.2.2. Since we now know that $\log c_d$ does not grow linearly, but its asymptotic behaviour depends on a subleading logarithmic term, we can include a $\log d$ term when fitting the linear regression. This means that we are fitting the following function

$$ad + b\log(d) + c$$

to our data $\{(d, \log(c_d)) \mid c_d \neq 0\}$. However, this results in a less accurate prediction. Using the formulae from Theorem 3.4.2, we expect to see the dimension as $-2b$. However, as

shown in Figure 3.22, the convergence is extremely slow, and we are nowhere near the correct dimension (five), even after computing $40\,000$ terms of the period sequence (without sampling).

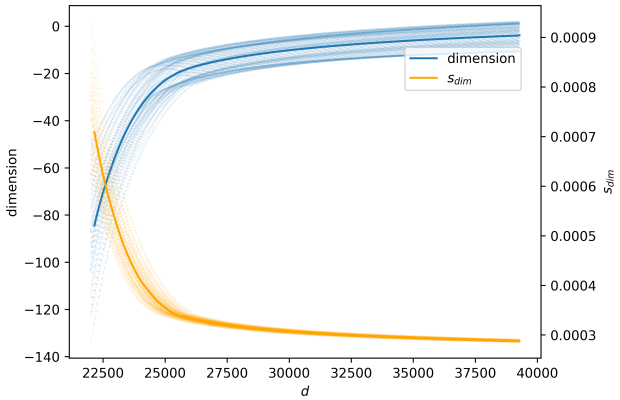


Figure 3.22: The dimension and its standard error (s_{dim}) for Example 3.2.2, computed from pairs $(k, \log c_k)$ such that $d - 20\,000 \leq k \leq d$ by sampling every 100th term, together with a LOWESS-smoothed trend line.

On the other hand, when we fit the logarithm of the period sequence with a linear regression and use the slope and the y-intercept to predict the dimension, we obtain a much better estimate for the dimension. In fact, if we use the regression data obtained by the last interval from Figure 3.9, it is incorrectly classified as a six-dimensional variety by the linear SVM, we have trained in Section 3.3.2; see Figure 3.23. While the predicted dimension is incorrect, while it is off only by one, the dimensions that result from Figure 3.22 are not even in the correct range.

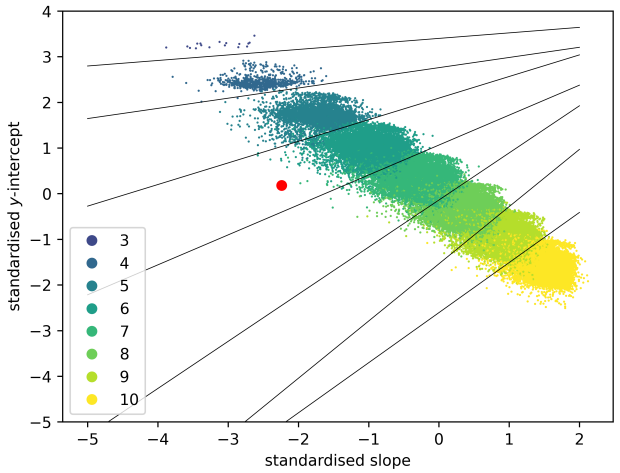


Figure 3.23: Figure 3.13 with a superimposed red dot, which correspond to the regression data computed in Example 3.2.2.

4 Asymptotics of the Regularized Quantum Period

In this chapter, we construct the asymptotics for the regularized quantum period of toric Fano varieties which are $\mathbb{Q}$-factorial. The result expands and supersedes the earlier result for toric Fano varieties with Picard rank one and two from [CKV23b], recalled in Section 3.4 (Theorem 3.4.1 and Theorem 3.4.2). We also discuss some work in progress for the construction in the context of the regularized quantum period of toric complete intersections.

4.1 Toric varieties

Let X be a $\mathbb{Q}$-factorial toric Fano variety of dimension D and Picard rank r. Assume this data is given as a weight matrix $W \in \mathbb{R}^{r\times N}$ (where $N = D + r$), as described in Section 1.1. Let us write the columns of W by $\alpha_1, \ldots, \alpha_N \in \mathbb{R}^r$ and $\alpha = \sum_{i=1}^N \alpha_i$. Recall that the regularized quantum period for X is expressed as

$$\hat{G}_X(t) = \sum_{k\in C\cap\mathbb{Z}^r} t^{\alpha^T\cdot k} \frac{(\alpha^T\cdot k)!}{\prod_{i=1}^N (\alpha_i^T\cdot k)!} \tag{4.1.1}$$

where C is the strictly convex cone in $\mathbb{R}^r$ defined by

$$C := \{x \in \mathbb{R}^r \mid \alpha_i^T \cdot x \geq 0 \text{ for all } i\} .$$

We want to study the asymptotic behaviour of c_d (the dth coefficient of the regularized quantum period $\hat{G}_X$) as $d \to \infty$. Fix $d \gg 0$, then

$$c_d = \sum_{\substack{k\in C\cap\mathbb{Z}^r \\ \alpha^T\cdot k = d}} \frac{d!}{\prod_{i=1}^N (\alpha_i^T\cdot k)!} .$$

We show that the summands are approximated by a rescaled Gaussian with the mean corresponding to $k^* \in \mathbb{Z}^r \cap C$ such that $\alpha^T \cdot k^* = d$ and c_{k^*} maximizes the summands. This

corresponds to finding k^* such that it minimises the quantity

$$\prod_{i=1}^{N}(\alpha_i^T \cdot k)!\,,$$

with constraint $\alpha^T \cdot k = d$. For $k \in C$ and $d \gg 0$, we can approximate this expression using Stirling's formula

$$\prod_{i=1}^{N}(\alpha_i^T \cdot k)! \sim (2\pi)^{N/2} e^{-\alpha^T \cdot k} \prod_{i=1}^{N}(\alpha_i^T \cdot k)^{\alpha_i^T \cdot k + 1/2}\,.$$

We can ignore the constants and just minimise

$$\prod_{i=1}^{N}(\alpha_i^T \cdot k)^{\alpha_i \cdot k}\,,$$

with constraint $\alpha^T \cdot k = d$. Note that we ignore the exponents $1/2$ since they do not matter as $d \to \infty$.

Proposition 4.1.1. *The constrained optimisation problem*

$$\min\left(\prod_{i=1}^{N}(\alpha_i^T \cdot x)^{\alpha_i^T \cdot x}\right) \qquad \textit{subject to } \begin{cases} x \in C^\circ \\ \alpha^T \cdot x = d \end{cases} \tag{4.1.2}$$

has a unique solution x^*, where

$$C = \{x \in \mathbb{R}^r \mid \alpha_i^T \cdot x \geq 0 \textit{ for all } i\}\,.$$

Furthermore, setting

$$p_i = \frac{\alpha_i^T \cdot x^*}{\alpha^T \cdot x^*} \quad \textit{for } i = 1, \ldots, N\,,$$

for $\alpha = \sum_{i=1}^{N} \alpha_i$, we have that

$$\prod_{i=1}^{N} p_i^{\alpha_i^T \cdot k}$$

depends on $k \in \mathbb{Z}^r$ only via $\alpha^T \cdot k$.

Proof. Taking logarithms gives the equivalent problem

$$\min \sum_{i=1}^{N}(\alpha_i^T \cdot x)\log(\alpha_i^T \cdot x) \qquad \text{subject to } \begin{cases} x \in C^\circ \\ \alpha^T \cdot x = d \end{cases}\,. \tag{4.1.3}$$

The objective function $\sum_{i=1}^{N}(\alpha_i^T \cdot x)\log(\alpha_i^T \cdot x)$ is the pullback to $\mathbb{R}^r$ of $f : \mathbb{R}^N \to \mathbb{R}$

$$f(x_1, \ldots, x_N) = \sum_{i=1}^{N} x_i \log x_i$$

along the linear embedding $\varphi : \mathbb{R}^r \to \mathbb{R}^N$ given by $(\alpha_1, \ldots, \alpha_N)$. Note that C is the preimage under φ of the positive orthant $\mathbb{R}^N_{\geq 0}$, so we need to minimise f on the intersection of the simplex

$$\{(x_1, \ldots, x_N) \in \mathbb{R}^N_{\geq 0} \mid x_1 + \cdots + x_N = d\}$$

with the image of φ. The function f is strongly convex (since it is the sum of strongly convex functions) and it decreases as we move away from the boundary of the simplex. Moreover, φ is injective, meaning that the composition $f \circ \varphi$ is strongly convex and therefore the minimisation problem in (4.1.3) has a unique solution x^* that lies in the strict interior of C. We can therefore find the global minimum x^* using the method of Lagrange multipliers, by solving

$$\sum_{i=1}^{N} \alpha_i \log(\alpha_i^T \cdot x) + \alpha = \lambda\alpha \tag{4.1.4}$$

for $\lambda \in \mathbb{R}$ and $x \in C^\circ$ with $\alpha^T \cdot x = d$. Thus,

$$\sum_{i=1}^{N} \alpha_i \log(\alpha_i^T \cdot x^*) = (\lambda - 1)\alpha$$

and, evaluating on $k \in \mathbb{Z}^r$ and exponentiating, we see that

$$\prod_{i=1}^{N} (\alpha_i^T \cdot x^*)^{\alpha_i^T \cdot k}$$

depends only on $\alpha^T \cdot k = d$. The result follows. □

The solution x^* from Proposition 4.1.1, can be calculated as the unique solution to the system of homogeneous real polynomial equations

$$\begin{cases} \prod_{i=1}^{N}(\alpha_i^T \cdot x)^{\alpha_i^1 \alpha^r} & = \prod_{i=1}^{N}(\alpha_i^T \cdot x)^{\alpha_i^r \alpha^1} \\ \vdots & \\ \prod_{i=1}^{N}(\alpha_i^T \cdot x)^{\alpha_i^{r-1} \alpha^r} & = \prod_{i=1}^{N}(\alpha_i^T \cdot x)^{\alpha_i^r \alpha^{r-1}} \end{cases}, \tag{4.1.5}$$

where $\alpha_i = (\alpha_i^1, \ldots, \alpha_i^r)$ and $\alpha = (\alpha^1, \ldots, \alpha^r)$. This is obtained by solving the appropriate Lagrange multiplier problem in(4.1.4). We will discuss methods of computing x^* efficiently in Section 4.3.

Theorem 4.1.2. *Let $(\alpha)_{i=1,\ldots,N} \in \mathbb{Z}^{r \times N}$ be an integer–valued weight matrix with columns α_i for a $\mathbb{Q}$-factorial toric Fano variety X of dimension D and Picard rank r ($N = D+r$). Then, as $d \to \infty$,*

$$\log(c_d) \sim Ad + B - \frac{D}{2}\log(d)\,,$$

where A and B are constants that depend only on the weight matrix.

Proof. The dth term of period sequence of X is computed as

$$c_d = \sum_{\substack{k\in\mathbb{Z}^r\cap C\\ \alpha^T\cdot k=d}} \frac{d!}{\prod_{i=1}^N(\alpha_i^T\cdot k)!}\,.$$

Fix $R>0$ and for $d\gg 0$ consider the ball $B_{R\sqrt{d}}$ around $x^*\in C^\circ$, the solution to the minimisation problem in Proposition 4.1.1. Then, we apply the *Local Theorem* from [Gne18], which we recall here.

Theorem 4.1.3. (Local Theorem, *Ch. 2 §12, [Gne18]*) *For $p_1,\dots,p_n>0$ such that $p_1+\cdots+p_n=1$, the ratio*

$$\frac{d!}{\prod_{i=1}^n k_i!}\prod_{i=1}^n p_i^{k_i}\,:\,\frac{\exp(-\frac{1}{2}\sum_{i=1}^n x_i^2)}{(2\pi d)^{\frac{n-1}{2}}\sqrt{p_1\cdots p_n}}\to 1$$

as $d\to\infty$, uniformly in all k_i's, where

$$x_i=\frac{k_i-dp_i}{\sqrt{dp_i}}$$

and the x_i's lie in bounded intervals.

Let

$$k_i:=\alpha_i^T\cdot k\,,\qquad p_i:=\tfrac{\alpha_i^T\cdot x^*}{\alpha^T\cdot x^*}\,,\qquad x_i:=\tfrac{k_i-dp_i}{\sqrt{dp_i}}\,,$$

where $k\in\mathbb{Z}^r\cap C\cap B_{R\sqrt{d}}$ and $\alpha^T\cdot k=\alpha^T\cdot x^*=d$. We can apply Theorem 4.1.3 since $k\in B_{R\sqrt{d}}$ ensures that the $x_i's$ lie in a bounded interval as $d\to\infty$. Given the solution x^* from Proposition 4.1.1, we can conclude that

$$\prod_{i=1}^N p_i^{k_i}=\prod_{i=1}^N p_i^{\alpha_i^T\cdot x^*}=\prod_{i=1}^N p_i^{dp_i}\,.$$

Moreover,

$$\sum_{i=1}^N x_i^2=\frac{(k-x^*)^T\mathcal{A}(k-x^*)}{d}$$

where $\mathcal{A}$ is the $r\times r$ matrix given by

$$\mathcal{A}=\sum_{i=1}^N\frac{1}{p_i}\alpha_i\alpha_i^T\,.$$

This matrix is positive-definite since it is equal to $H_f(x^*)$, the Hessian of $f:\mathbb{R}^N\to\mathbb{R}$ defined as

$$x\mapsto\sum_{i=1}^N(\alpha_i^T\cdot x)\log\left(\alpha_i^T\cdot x\right)\,,$$

which we have said is a strongly convex function in the proof of Proposition 4.1.1.

Therefore, the statement Theorem 4.1.3 applied to this case translates to saying that as $d \to \infty$ and for $k \in \mathbb{Z}^r \cap C \cap B_{R\sqrt{d}}$, such that $\alpha^T \cdot k = d$, we have

$$c_k = \frac{d!}{\prod_{i=1}^N (\alpha_i^T \cdot k)!} \sim \frac{1}{(2\pi d)^{\frac{N-1}{2}} \prod_{i=1}^N p_i^{dp_i+\frac{1}{2}}} \exp\left(-\frac{(k-x^*)^T \mathcal{A}(k-x^*)}{2d}\right) .$$

Note that, for $d \gg 0$, those c_k for $k \in \mathbb{Z}^r \cap C$, $\alpha^T \cdot k = d$, but $k \notin B_{R\sqrt{d}}$, carry a negligible contribution, therefore,

$$c_d \sim \frac{1}{(2\pi d)^{\frac{N-1}{2}} \prod_{i=1}^N p_i^{dp_i+\frac{1}{2}}} \sum_{\substack{k \in \mathbb{Z}^r \cap C \\ \alpha^T \cdot k = d}} \exp\left(-\frac{(k-x^*)^T \mathcal{A}(k-x^*)}{2d}\right) .$$

Applying the change of co-ordinate $y = (k - x^*)/\sqrt{d}$, and considering the sum here as a Riemann sum we obtain

$$c_d \sim \frac{1}{(2d\pi)^{\frac{N-1}{2}} \prod_{i=1}^N p_i^{dp_i+\frac{1}{2}}} d^{\frac{r-1}{2}} \int_{y \in L_\alpha} \exp\left(-\frac{1}{2} y^T \mathcal{A} y\right) \mathrm{d}y$$

where L_α is the linear subspace given by $\ker(\alpha)$ and $\mathrm{d}y$ is the measure on L_α given by the integer lattice $\mathbb{Z}^r \cap L_\alpha \subset L_\alpha$. In order to evaluate the integral, let $\mathcal{B}$ be the $(r-1) \times r$ matrix whose columns form a $\mathbb{Z}$-basis of L_α. Observe that the pull-back of $\mathrm{d}y$ along the map $\mathbb{R}^{r-1} \to L_\alpha$ given by $t \mapsto \mathcal{B}t$ is the standard measure on $\mathbb{R}^{r-1}$. Thus,

$$\int_{L_\alpha} \exp\left(-\frac{1}{2} y^T \mathcal{A} y\right) \mathrm{d}y = \int_{\mathbb{R}^r} \exp\left(-\frac{1}{2} x^T \mathcal{C} x\right) \mathrm{d}x = \sqrt{\frac{(2\pi)^{r-1}}{\det \mathcal{C}}} ,$$

where $\mathcal{C} = \mathcal{B} \cdot \mathcal{A} \cdot \mathcal{B}^T$.

Therefore,

$$c_d \sim \frac{1}{(2d\pi)^{\dim X/2} \prod_{i=1}^N p_i^{dp_i+1/2} \sqrt{\det(\mathcal{C})}}$$

since $\dim X = D = N - r$ □

Remark 4.1.4. The following is a concrete, co-ordinate dependent, definition of the matrix $\mathcal{C}$ from the proof,

$$\mathcal{C}_{ij} = \sum_{k=1}^N \frac{(\alpha^r \alpha_k^j - \alpha^j \alpha_k^r)(\alpha^r \alpha_k^i - \alpha^i \alpha_k^r)}{l^r p_k (\alpha^r)^{\frac{r-2}{r}}} ,$$

where $l = \gcd(\alpha^1, \ldots, \alpha^r)$, and $\alpha_i = (\alpha_i^1, \ldots, \alpha_i^r)$, $\alpha = (\alpha^1, \ldots, \alpha^r)$. This is obtained by projecting the co-ordinates $k = (k_1, \ldots, k_r) \in \mathbb{Z}^r$ on the $\{k_r = 0\}$ co-ordinate hyperplane, and computing the integral over the r-simplex in $\mathbb{R}^{r-1}$

$$\Delta_d := \{x \in \mathbb{R}^{n-1}_{\geq 0} \mid \alpha^T \cdot x \leq d\} .$$

4.2 Toric complete intersections

In this section we discuss current progress in generalising this asymptotic result to the setting of toric complete intersections. Recall from Section 1.5 that we encode the data defining a toric complete intersection in the following way. Let X be a $\mathbb{Q}$-factorial toric variety given by an $r \times N$ weight matrix with columns $(\alpha_i)_{i=1,\dots,N}$. Then let Y be the quasi-smooth complete intersection in X defined by a section of the bundle $E = L_1 \oplus \cdots \oplus L_M$, where each L_i is convex. Assume Y is Fano and an orbifold. Let $\beta_i = c_1(L_i)$, this data can also be written in an $r \times M$ matrix with columns $(\beta)_{i=1,\dots,M}$. Note that the β_i's belong to the cone generated by the α_i's. We assume Y to be Fano, meaning that $w = \sum_{i=1}^{N} \alpha_i - \sum_{i=1}^{M} \beta_i$ lives in the ample cone of the toric complete intersection.

Remark 4.2.1. Assuming X is Fano and taking the empty complete intersection, the above set up recovers the Fano toric case from Section 4.1.

Let C be the convex cone

$$C = \{x \in \mathbb{R}^r \mid \alpha_i^T \cdot x \geq 0 \text{ for all } i\} .$$

We will consider the regularized version of the following hypergeometric series

$$G_X(t) = e^{-ct} \sum_{d \geq 0} \left(\sum_{\substack{k \in C \cap \mathbb{Z}^r \\ \omega^T \cdot k = d}} \frac{\prod_{i=1}^{M} (\beta_i^T \cdot k)!}{\prod_{i=1}^{N} (\alpha_i^T \cdot k)!} \right) t^d \tag{4.2.1}$$

which, if $p_d = d! \frac{\prod_{i=1}^{M} (\beta_i^T \cdot k)!}{\prod_{i=1}^{N} (\alpha_i^T \cdot k)!}$, then $c = p_1$ and the regularized version of $G_X(t)$ is

$$\hat{G}_X(t) = \sum_{d \geq 0} \sum_{k=0}^{d} \binom{d}{k} (-c)^{d-k} p_k t^d . \tag{4.2.2}$$

For smooth toric complete intersections this is proven to be their regularized quantum period [CCGK16]. This generalises to a wider class of examples, namely Fano toric complete intersections which are quasi-smooth and have an orbifold structure (see Section 1.5 for details). We do not discuss the details here, but just study the asymptotic behaviour of the coefficients of this hypergeometric series.

Conjecture 4.2.2. *Given data describing a quasi-smooth Fano toric complete intersection as above, and c_d the coefficients of $\hat{G}_X(t)$ as in* (4.2.2)*, then*

$$\log(c_d) \sim Ad - \frac{D}{2} \log(d) + B$$

as $d \to \infty$, where $A, B \in \mathbb{R}$ are constants depending entirely on the initial geometric data.

In order to prove this in the same fashion as Theorem 4.1.2 for toric complete intersections we need an equivalent to Proposition 4.1.1 to hold. In order for that to be the case we need to impose the following assumption (which is always satisfied by $\mathbb{Q}$-factorial toric varieties, as seen in Section 4.1).

Assumption 4.2.3. Consider matrices $(\alpha_i)_{i=1,\dots,N} \in \mathbb{Z}^{r\times N}$ and $(\beta_i)_{i=1,\dots,M} \in \mathbb{Z}^{r\times M}$. The function $f : \mathbb{R}^r \to \mathbb{R}$

$$x \mapsto \sum_{i=1}^{N} (\alpha_i^T \cdot x) \log(\alpha_i^T \cdot x) - \sum_{i=1}^{M} (\beta_i^T \cdot x) \log(\beta_i^T \cdot x) \,, \tag{4.2.3}$$

is strictly convex on

$$C = \{x \in \mathbb{R}^r \mid \alpha_i^T \cdot x \geq 0 \text{ for all } i\} \,. \tag{4.2.4}$$

Lemma 4.2.4. *Let $\mathbb{P}(a_0, \dots, a_N)$ be a weighted projective space and consider the weighted projective intersection Y given by a section of $E = \mathcal{O}(d_1) \oplus \cdots \oplus \mathcal{O}(d_M)$, for $a_i, d_i \in \mathbb{Z}_{>0}$. Then, Assumption 4.2.3 holds if and only if Y is Fano.*

Proof. The function from Assumption 4.2.3 is $f : \mathbb{R} \to \mathbb{R}$ such that

$$\begin{aligned} f(x) &= \sum_{i=0}^{N} (a_i x) \log(a_i x) - \sum_{i=1}^{N} (d_i x) \log(d_i x) \\ &= x \log(x) \left(\sum_{i=0}^{N} a_i - \sum_{i=1}^{M} d_i \right) + x \left(\sum_{i=0}^{N} a_i \log(a_i) - \sum_{i=1}^{M} d_i \log(d_i) \right) \,. \end{aligned}$$

We note that f is strongly convex on $\mathbb{R}_{\geq 0}$ if and only if $x \log(x) \left(\sum_{i=0}^{N} a_i - \sum_{i=1}^{M} d_i \right)$ is, since the other term is linear in x. Clearly, f is strongly convex on $\mathbb{R}_{\geq 0}$ if and only if

$$\sum_{i=0}^{N} a_i - \sum_{i=1}^{M} d_i > 0 \,,$$

which holds if and only if Y is Fano. □

Lemma 4.2.5. *Assumption 4.2.3 holds for data $(\alpha_i)_{i=1,\dots,N}$ and $(\beta_i)_{i=1,\dots,M}$ for which there exists a partition of $\{1, \dots, N\}$, $\{S_0, S_1, \dots, S_M\}$ such that $\beta_i = \sum_{j\in S_i} \alpha_j$ for all $i = 1, \dots, M$.*

Remark 4.2.6. This is a case of interest since it is the same as the assumptions needed for the Givental/Hori-Vafa method to construct a Laurent polynomial mirror to a toric complete intersection to apply [Giv98, HKK+03]. Note that $S_0 \neq \varnothing$ is essential.

Proof. The function from Assumption 4.2.3 is $f : \mathbb{R}^r \to \mathbb{R}$ such that

$$f(x) = \sum_{i=1}^{N} (\alpha_i^T \cdot x) \log\left(\alpha_i^T \cdot x\right) - \sum_{i=1}^{M} (\beta_i^T \cdot x) \log\left(\beta_i^T \cdot x\right) \,,$$

Note that f can be written as $\sum_{j=0}^{M} f_{S_j}$ using the partition $(S_0, S_1, \ldots, S_M)$, where

$$f_{S_0}(x) := \sum_{i \in S_0} (\alpha_i^T \cdot x) \log\left(\alpha_i^T \cdot x\right),$$

$$f_{S_j}(x) := \sum_{i \in S_j} (\alpha_i^T \cdot x) \log\left(\alpha_i^T \cdot x\right) - \left(\sum_{i \in S_j} (\alpha_i^T \cdot x)\right) \log\left(\sum_{i \in S_j} (\alpha_i^T \cdot x)\right) \text{ for } j = 1, \ldots, M.$$

Note that $f_{S_0}(x)$ is the sum of strongly convex functions, and hence it is strongly convex. We show that for each $j = 1, \ldots, M$, f_{S_j} is convex, and therefore f is strongly convex, concluding the proof. For $j = 1, \ldots, M$ the Hessian of f_{S_j} is

$$\begin{aligned} H_{f_{S_j}}(x) &= \sum_{i \in S_j} \frac{\alpha_i \cdot \alpha_i^T}{\alpha_i^T \cdot x} - \frac{\beta_j \cdot \beta_j^T}{\beta_j^T \cdot x} = \sum_{i \in S_j} \frac{\alpha_i \cdot \alpha_i^T}{\alpha_i^T \cdot x} - \frac{\left(\sum_{i \in S_j} \alpha_i\right) \cdot \left(\sum_{i \in S_j} \alpha_i\right)^T}{\sum_{i \in S_j} \alpha_i^T \cdot x} \\ &= \frac{\sum_{\substack{i,k \in S_j \\ i<k}} (\alpha_i(\alpha_k^T \cdot x) + \alpha_k(\alpha_i^T \cdot x))^2 \prod_{\substack{l \in S_j \\ l \neq i,k}} (\alpha_l^T \cdot x)}{\prod_{i \in S_j} (\alpha_i^T \cdot x) \left(\sum_{i \in S_j} (\alpha_i^T \cdot x)\right)} \end{aligned}$$

which is non-negative definite for $x \in C$, so f_{S_j} is strongly convex. □

Example 4.2.7. For weighted complete intersections, Lemma 4.2.4 shows that Assumption 4.2.3 holds if and only if the variety is Fano. However, these conditions are not equivalent for general toric complete intersections, as we will now show.
Consider the three-dimensional toric variety X of Picard rank two given by the weight matrix

$$\begin{bmatrix} 1 & 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 3 & 1 \end{bmatrix}.$$

Consider the toric complete intersection Y given by the divisor of bi-degree $(2, 4)$. Therefore, in our notation $\beta_1 = \binom{2}{4}$. Then, we note that $\omega = \binom{3}{2}$ which does not belong to the ample cone of the toric ambient space[13], which coincides with the ample cone of the toric complete intersection since the chosen line bundle is ample. Therefore, the toric complete intersection is not Fano, however Assumption 4.2.3 holds by Lemma 4.2.5, since

$$\binom{2}{4} = \binom{1}{1} + \binom{1}{3}.$$

When Assumption 4.2.3 holds we have the following result which is the equivalent to Proposition 4.1.1 in the toric complete intersection case.

Proposition 4.2.8. *If Assumption 4.2.3 holds, the constrained optimisation problem*

$$\min\left(\frac{\prod_{i=1}^{N} (\alpha_i^T \cdot x)^{\alpha_i^T \cdot x}}{\prod_{i=1}^{M} (\beta_i^T \cdot x)^{\beta_i^T \cdot x}}\right) \qquad \textit{subject to } \begin{cases} x \in C^\circ \\ \omega^T \cdot x = d \end{cases} \tag{4.2.5}$$

[13]The ample cone is the cone where the sum of the weight matrix columns $\binom{5}{6}$ lives, i.e. the cone Cone $\left(\binom{1}{1}, \binom{1}{3}\right)$.

has a unique solution x^*, for

$$C = \{x \in \mathbb{R}^r \mid \alpha_i^T \cdot x \geq 0 \text{ for all } i\} .$$

Furthermore, setting

$$\begin{aligned} p_i &= \frac{\alpha_i^T \cdot x^*}{\omega^T \cdot x^*} \quad \text{for } i = 1, \dots, N , \\ q_i &= \frac{\beta_i^T \cdot x^*}{\omega^T \cdot x^*} \quad \text{for } i = 1, \dots, M , \end{aligned}$$

for $\omega = \sum_{i=1}^N \alpha_i - \sum_{i=1}^M \alpha_i$, we have that

$$\frac{\prod_{i=1}^N p_i^{\alpha_i^T \cdot k}}{\prod_{i=1}^M q_i^{\beta_i^T \cdot k}}$$

depends on $k \in \mathbb{Z}^r$ only via $\omega^T \cdot k$.

Proof. The proof is equivalent to the proof of Proposition 4.1.1, using explicitly that the global minimum exists because the function $f : \mathbb{R}^r \to \mathbb{R}$

$$x \mapsto \sum_{i=1}^N (\alpha_i^T \cdot x) \log(\alpha_i^T \cdot x) - \sum_{i=1}^M (\beta_i^T \cdot x) \log(\beta_i^T \cdot x)$$

is assumed to be strongly convex by Assumption 4.2.3. □

As for the toric case, the solution x^* to Proposition 4.2.8 can be calculated as the solution to the homogeneous system of polynomial equations

$$\begin{cases} \prod_{i=1}^N (\alpha_i^T \cdot x)^{\alpha_i^1 \omega^r} \prod_{i=1}^M (\beta_i^T \cdot x)^{\beta_i^r \omega^1} & = \prod_{i=1}^N (\alpha_i^T \cdot x)^{\alpha_i^r \omega^1} \prod_{i=1}^M (\beta_i^T \cdot x)^{\beta_i^1 \omega^r} \\ \vdots & \\ \prod_{i=1}^N (\alpha_i^T \cdot x)^{\alpha_i^{r-1} \omega^r} \prod_{i=1}^M (\beta_i^T \cdot x)^{\beta_i^r \omega^{r-1}} & = \prod_{i=1}^N (\alpha_i^T \cdot x)^{\alpha_i^r \omega^{r-1}} \prod_{i=1}^M (\beta_i^T \cdot x)^{\beta_i^{r-1} \omega^r} \end{cases} \tag{4.2.6}$$

where $\alpha_i = (\alpha_i^1, \dots, \alpha_i^r)$, $\beta_i = (\beta_i^1, \dots, \beta_i^r)$, $\omega = (\omega^1, \dots, \omega^r)$.

We prove Conjecture 4.2.2 in the case of a toric complete intersection that satisfy Assumption 4.2.3 and such that $c = 0$ in (4.2.2). In this case the regularized quantum period is

$$\hat{G}_X(t) = \sum_{d \geq 0} \left(\sum_{\substack{k \in C \cap \mathbb{Z}^r \\ \omega^T \cdot k = d}} \frac{\prod_{i=1}^M (\beta_i^T \cdot k)!}{\prod_{i=1}^N (\alpha_i^T \cdot k)!} \right) d! t^d .$$

For this we need a generalisation of Theorem 4.1.3 to this setting.

Theorem 4.2.9. *For $p_1, \dots, p_n, q_1, \dots, q_m > 0$ such that $p_1 + \cdots + p_n - q_1 - \cdots - q_m = 1$, the ratio*

$$d! \frac{\prod_{i=1}^m l_i!}{\prod_{i=1}^n k_i!} \prod_{i=1}^n p_i^{k_i} \prod_{i=1}^m q_i^{-l_i} \ : \ \frac{\exp(-\frac{1}{2}\sum_{i=1}^n x_i^2 + \frac{1}{2}\sum_{i=1}^m y_i^2)\sqrt{q_1 \cdots q_m}}{(2\pi d)^{\frac{n-m-1}{2}} \sqrt{p_1 \cdots p_n}} \to 1$$

as $d \to \infty$, uniformly in all k_i's and all l_i's, where

$$x_i = \frac{k_i - dp_i}{\sqrt{dp_i}} \qquad\qquad y_i = \frac{l_i - dq_i}{\sqrt{dq_i}}$$

and the x_i's and the y_i's lie in bounded intervals.

Proof. This proof is adapted from the proof of De Moivre–Laplace theorem from Theorem 5 §7.3 [CA03]. Using Stirling's formula repeatedly, we can approximate the following,

$$\begin{aligned} d! \frac{\prod_{i=1}^m l_i!}{\prod_{i=1}^n k_i!} \prod_{i=1}^n p_i^{k_i} \prod_{i=1}^m q_i^{-l_i} &\sim \frac{d^{d+\frac{1}{2}} (2\pi)^{\frac{m+1}{2}} \prod_{i=1}^m l_i^{l_i+\frac{1}{2}}}{(2\pi)^{\frac{n}{2}} \prod_{i=1}^n k_i^{k_i+\frac{1}{2}}} \prod_{i=1}^n p_i^{k_i} \prod_{i=1}^m q_i^{-l_i} \\ &= \frac{1}{(2d\pi)^{\frac{n-m-1}{2}}} \sqrt{\frac{\prod_{i=1}^m l_i/d}{\prod_{i=1}^n k_i/d}} \prod_{i=1}^n \left(\frac{dp_i}{k_i}\right)^{k_i} \prod_{i=1}^m \left(\frac{l_i}{dq_i}\right)^{l_i} \\ &\sim \frac{1}{(2d\pi)^{\frac{n-m-1}{2}}} \sqrt{\frac{\prod_{i=1}^m p_i}{\prod_{i=1}^n q_i}} \prod_{i=1}^n \left(\frac{dp_i}{k_i}\right)^{k_i} \prod_{i=1}^m \left(\frac{l_i}{dq_i}\right)^{l_i} . \end{aligned}$$

Here the last approximation is justified by, since the x_i's and the y_i's lie in bounded intervals, we have that $\frac{k_i}{d} \to p_i$ and $\frac{l_i}{d} \to q_i$ as $d \to \infty$.

Moreover, let

$$x_i = \frac{k_i - dp_i}{\sqrt{dp_i}} \quad \text{for } i = 1, \dots, n \quad \text{and} \quad y_i = \frac{l_i - dq_i}{\sqrt{dq_i}} \quad \text{for } i = 1, \dots, m .$$

Then,

$$\begin{aligned} &\prod_{i=1}^n \left(\frac{dp_i}{k_i}\right)^{k_i} \prod_{i=1}^m \left(\frac{l_i}{dq_i}\right)^{l_i} = \exp\left[\sum_{i=1}^m l_i \log\left(\frac{l_i}{dq_i}\right) - \sum_{i=1}^n k_i \log\left(\frac{k_i}{dp_i}\right)\right] \\ &= \exp\left[\sum_{i=1}^m (dq_i + y_i\sqrt{dq_i}) \log\left(1 + \frac{y_i\sqrt{dq_i}}{dq_i}\right) - \sum_{i=1}^n (dp_i + x_i\sqrt{dp_i}) \log\left(1 + \frac{x_i\sqrt{dp_i}}{dp_i}\right)\right] \\ &\sim \exp\left[\sum_{i=1}^m (dq_i + y_i\sqrt{dq_i}) \left(\frac{y_i\sqrt{dq_i}}{dq_i} - \frac{y_i^2}{2dq_i} \cdot \cdot\right) - \sum_{i=1}^n (dp_i + x_i\sqrt{dp_i}) \left(\frac{x_i\sqrt{dp_i}}{dp_i} - \frac{x_i^2}{2dp_i} \cdot \cdot\right)\right] \\ &= \exp\left[\sum_{i=1}^m (y_i\sqrt{dq_i} + \frac{y_i^2}{2} \cdots) - \sum_{i=1}^n (x_i\sqrt{dp_i} + \frac{x_i^2}{2} \cdots)\right] \sim \exp\left[-\frac{1}{2}\left(\sum_{i=1}^n x_i^2 - \sum_{i=1}^m y_i^2\right)\right] . \end{aligned}$$

Here we are using the approximation $\log(1+x) \sim x - \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$, which converges since we are assuming the x_i's and the y_i's lie in bounded intervals. □

Theorem 4.2.10. *Consider matrices $(\alpha_i)_{i=1,\dots,N} \in \mathbb{Z}^{r\times N}$ and $(\beta_i)_{i=1,\dots,M} \in \mathbb{Z}^{r\times M}$, such that Assumption 4.2.3 holds. Then, consider the hypergeometric sequence*

$$c_d = \sum_{\substack{k\in C\cap\mathbb{Z}^r \\ \omega^T\cdot k=d}} \frac{\prod_{i=1}^M(\beta_i^T\cdot k)!}{\prod_{i=1}^N(\alpha_i^T\cdot k)!} d!$$

where $\omega = \sum_{i=1}^N \alpha_i - \sum_{i=1}^M \beta_i$ and

$$C = \{x \in \mathbb{R}^r \mid \alpha_i^T \cdot x \geq 0 \text{ for all } i\} .$$

Then

$$\log(c_d) \sim Ad - \frac{N-M-r}{2}\log(d) + B$$

as $d \to \infty$, where $A, B \in \mathbb{R}$ are constants that depend only on $(\alpha_i)_{i=1,\dots,N}$ and $(\beta_i)_{i=1,\dots,M}$.

Remark 4.2.11. If the matrix-data in Theorem 4.2.10 gives rise to a toric complete intersection X, then the asymptotic formula is equivalent to the one for toric varieties in Theorem 4.1.2, since in this case $\dim(X) = \frac{N-M-r}{2}$.

Proof. The proof follows the same steps as the proof of Theorem 4.1.2 with some minor changes that we highlight below. Consider

$$c_d = \sum_{\substack{k\in\mathbb{Z}^r\cap C \\ \omega^T\cdot k=d}} \frac{\prod_{i=1}^M(\beta_i^T\cdot k)!}{\prod_{i=1}^N(\alpha_i^T\cdot k)!} d! \, .$$

Fix $R > 0$ and for $d \gg 0$ consider the ball $B_{R\sqrt{d}}$ around $x^* \in C^\circ$, the solution to the minimisation problem in Proposition 4.2.8. Then, we apply Theorem 4.2.9, setting

$$\begin{aligned} k_i &:= \alpha_i^T\cdot k\,, & p_i &:= \tfrac{\alpha_i^T\cdot x^*}{\omega^T\cdot x^*}\,, & x_i &:= \tfrac{k_i - dp_i}{\sqrt{dp_i}}\,, \\ l_i &:= \beta_i\cdot k\,, & q_i &:= \tfrac{\beta_i^T\cdot x^*}{\omega^T\cdot x^*}\,, & y_i &:= \tfrac{l_i - dq_i}{\sqrt{dq_i}}\,, \end{aligned}$$

where $k \in \mathbb{Z}^r \cap C \cap B_{R\sqrt{d}}$ and $\omega^T\cdot k = \omega^T\cdot x^* = d$. We can apply Theorem 4.2.9 since $k \in B_{R\sqrt{d}}$ ensures that the $x_i's$ and y_i's lie in a bounded interval as $d \to \infty$. Given the solution x^* from Proposition 4.2.8, we can conclude that

$$\begin{aligned} \prod_{i=1}^N p_i^{k_i} &= \prod_{i=1}^N p_i^{\alpha_i^T\cdot x^*} = \prod_{i=1}^N p_i^{dp_i}\,, \\ \prod_{i=1}^M q_i^{k_i} &= \prod_{i=1}^M q_i^{\beta_i^T\cdot x^*} = \prod_{i=1}^M q_i^{dq_i}\,. \end{aligned}$$

Moreover,

$$\sum_{i=1}^N x_i^2 - \sum_{i=1}^M y_i^2 = \frac{(k-x^*)^T\mathcal{A}(k-x^*)}{d}$$

where $\mathcal{A}$ is the $r \times r$ matrix given by

$$\mathcal{A} = \sum_{i=1}^{N} \frac{1}{p_i} \alpha_i \alpha_i^T - \sum_{i=1}^{M} \frac{1}{q_i} \beta_i \beta_i^T \, .$$

This matrix is positive-definite since it is equal to $H_f(x^*)$, the Hessian of $f : \mathbb{R}^{N+M} \to \mathbb{R}$ defined as

$$x \mapsto \sum_{i=1}^{N} (\alpha_i^T \cdot x) \log\left((\alpha_i^T \cdot x)\right) - \sum_{i=1}^{M} (\beta_i^T \cdot x) \log\left((\beta_i^T \cdot x)\right) \, ,$$

which is strictly convex by Assumption 4.2.3.

Therefore, Theorem 4.2.9 applied to this case translates to saying that as $d \to \infty$ and for $k \in \mathbb{Z}^r \cap C \cap B_{R\sqrt{d}}$, such that $\omega^T \cdot k = d$, we have

$$d! \frac{\prod_{i=1}^{M} (\beta_i^T \cdot k)!}{\prod_{i=1}^{N} (\alpha_i^T \cdot k)!} \sim \frac{\prod_{i=1}^{M} q_i^{dq_i + \frac{1}{2}}}{(2\pi d)^{\frac{N-M-1}{2}} \prod_{i=1}^{N} p_i^{dp_i + \frac{1}{2}}} \exp\left(- \frac{(k - x^*)^T \mathcal{A} (k - x^*)}{2d} \right) \, .$$

Note that, for $d \gg 0$, those c_k for $k \in \mathbb{Z}^r \cap C$, $\omega^T \cdot k = d$, but $k \notin B_{R\sqrt{d}}$, carry a negligible contribution, therefore,

$$c_d \sim \frac{\prod_{i=1}^{M} q_i^{dq_i + \frac{1}{2}}}{(2\pi d)^{\frac{N-M-1}{2}} \prod_{i=1}^{N} p_i^{dp_i + \frac{1}{2}}} \sum_{\substack{k \in \mathbb{Z}^r \cap C \\ \omega^T \cdot k = d}} \exp\left(- \frac{(k - x^*)^T \mathcal{A} (k - x^*)}{2d} \right) \, .$$

Applying the change of co-ordinate $y = (k - x^*)/\sqrt{d}$, and considering the sum here as a Riemann sum we have

$$c_d \sim \frac{\prod_{i=1}^{M} q_i^{dq_i + \frac{1}{2}}}{(2d\pi)^{\frac{N-M-1}{2}} \prod_{i=1}^{N} p_i^{dp_i + \frac{1}{2}}} d^{\frac{r-1}{2}} \int_{y \in L_\omega} \exp\left(- \frac{1}{2} y^T \mathcal{A} y \right) \mathrm{d}y$$

where L_ω is the linear subspace given by $\ker(\omega)$ and $\mathrm{d}y$ is the measure on L_ω given by the integer lattice $\mathbb{Z}^r \cap L_\omega \subset L_\omega$. In order to evaluate the integral, let $\mathcal{B}$ be the $(r-1) \times r$ matrix whose columns form a $\mathbb{Z}$-basis of L_ω. Observe that the pull-back of $\mathrm{d}y$ along the map $\mathbb{R}^{r-1} \to L_\omega$ given by $t \mapsto \mathcal{B}t$ is the standard measure on $\mathbb{R}^{r-1}$. Thus,

$$\int_{L_\omega} \exp\left(- \frac{1}{2} y^T \mathcal{A} y \right) \mathrm{d}y = \int_{\mathbb{R}^r} \exp\left(- \frac{1}{2} x^T \mathcal{C} x \right) \mathrm{d}x = \sqrt{\frac{(2\pi)^{r-1}}{\det \mathcal{C}}} \, ,$$

where $\mathcal{C} = \mathcal{B} \cdot \mathcal{A} \cdot \mathcal{B}^T$.

Therefore,

$$c_d \sim \frac{\prod_{i=1}^{M} q_i^{dq_i + \frac{1}{2}}}{(2d\pi)^{\frac{N-M-r}{2}} \prod_{i=1}^{N} p_i^{dp_i + 1/2} \sqrt{\det(\mathcal{C})}} \, .$$

□

Remark 4.2.12. As in Remark 4.1.4 we give a co-ordinate dependent definition of $\mathcal{C}$ which is useful in computations. It is obtained by the same projection of $k = (k_1, \ldots, k_r) \in \mathbb{Z}^r$ on

the $\{k_r = 0\}$ co-ordinate hyperplane.

$$\mathcal{C}_{ij} = \sum_{k=1}^{N} \frac{(\omega^r \alpha_k^j - \omega^j \alpha_k^r)(\omega^r \alpha_k^i - \omega^i \alpha_k^r)}{l^r p_k(\omega^r)^{\frac{r-2}{r}}} - \sum_{k=1}^{M} \frac{(\omega^r \beta_k^j - \omega^j \beta_k^r)(\omega^r \beta_k^i - \omega^i \beta_k^r)}{l^r q_k(\omega^r)^{\frac{r-2}{r}}}, \tag{4.2.7}$$

where $l = \gcd(\omega^1, \ldots, \omega^r)$, and $\alpha_i = (\alpha_i^1, \ldots, \alpha_i^r)$, $\beta_i = (\beta_i^1, \ldots, \beta_i^r)$, $\omega = (\omega^1, \ldots, \omega^r)$.

We expect Conjecture 4.2.2 to hold for the coefficients of the regularized quantum period when $c \neq 0$ in Eq (4.2.2) if we still assume Assumption 4.2.3.

4.3 Computing x^*

To compute x^* from Proposition 4.1.1 and 4.2.8, we can use constrained gradient descent, whose steps are made explicit in Algorithm 1. The input is assumed to be the data of the ambient toric variety (given as a list of columns α_i's of the weight matrix for the toric ambient), the data for the line bundles (given as a list of the β_i's, which in the toric case is just empty), the learning rate η, and the maximum number of iterations `iters`. A Python implementation of this algorithm is found in the repository [Ven24b]. The algorithm assumes and does not check that the function f from (4.2.3) is strongly convex.

Algorithm 1: The algorithm computes x^* as in Proposition 4.1.1 and Proposition 4.2.8.

Function `xStar`($(\alpha_i)_{i=1}^N$: *list of vectors*, $(\beta)_{i=1}^M$: *list of vectors*, η: *float*, `iters`: *int*): *vector* **is**
  $F \leftarrow$the function to minimise as in (4.2.3);
  $C \leftarrow$ the cone as in (4.2.4);
  $\omega \leftarrow \sum_{i=1}^N \alpha_i - \sum_{j=1}^M \beta_j$;
  $x_0 \leftarrow \frac{\omega}{\omega^T \cdot \omega}$ # Initial guess;
  for $n = 1, \ldots,$ `iters` **do**
    Compute $y = \nabla F(x_0)$;
    Take gradient step $x_1 = x_0 - \eta y$;
    Project on x_1 on the plane $\omega^T \cdot x = 1$;
    while $F(x_1) > F(x_0)$ *or* $x_1 \notin C$ **do**
      $x_1 \leftarrow (x_1 + x_0)/2$;
    end
    $x_0 \leftarrow x_1$;
  end
  Return x_0.
end

Remark 4.3.1. The existence and uniqueness of x^* in C assures that Algorithm 1 converges to a good approximation of x^* for a high enough number of iterations.

Note that in the case of Picard rank two toric varieties and toric complete intersections in Picard rank two toric varieties, the systems of homogeneous polynomial equations (4.1.5)

and (4.2.6) reduce to solving one single variable polynomial equation. In these cases, the solution $x^* = [\mu : \nu] \in \mathbb{P}^1$, so we can fix $\mu = 1$ and solve for ν (or the other way around). The domain for ν is determined by the cone, since

$$\begin{pmatrix} \mu \\ \nu \end{pmatrix} \in C$$

for C as in Propositions 4.1.1 and 4.2.8. Such univariate polynomial equations can be solved using numerical methods: this is how x^* was calculated to produce Figure 3.20 and how we will calculate x^* to construct the dataset in Section 5.3.1 in the next chapter. However, using Algorithm 1, we can compute the rigorous asymptotic data for varieties irrespective their Picard rank.

5 Detecting Terminal Singularities

Recall, in Chapter 3 we generated all terminal weighted projective spaces with dimension bounded by ten and weights bounded by 25. These were visualised using the coefficients (A, B) from Theorem 3.4.1 in Figure 3.18, which displayed a lot of geometric structure. We would like to conduct a similar study for toric varieties with higher Picard rank, however checking terminality in these cases is a computationally expensive problem in convex geometry. In Picard rank one (weighted projective spaces) there exists a fast combinatorial criterion that we have recalled in Proposition 1.1.3 – which is how we produced the data displayed in Figure 3.18. In this chapter, we investigate how to check the condition of being terminal for a Picard rank two toric variety using machine learning, with the aim of avoiding having to perform expensive computer algebra routines to produce a similar figure.

In this chapter we cover parts of the content of [CKV24].

5.1 Data generation

In this section, we describe the dataset used in the construction of our machine learning model. The datasets are generated using Magma v2.25-4, [BCP97].

The data is a balanced labelled dataset of 5 million terminal and 5 million non-terminal $\mathbb{Q}$-factorial toric varieties of Picard rank two and dimension eight. The motivation for the choice of dimension is two-fold. It is important to distance ourselves from the surface case, where the terminality check is trivial (recall that smooth implies terminal in dimension two). Moreover, we need to consider examples of sufficiently high dimension where there are an abundance of examples, so that we can benefit from a machine learning approach. For example, the analogue of our dataset in dimension three contains only 34 examples; see [Kas06]. On the other hand, the choice of Picard rank two is natural. As we have mentioned above, there already exists a fast combinatorial formula to check terminality in Picard rank one (Proposition 1.1.3), so the next step to consider is Picard rank two.

As in Chapter 3 we use generate toric Fano varieties using weight matrices. Therefore, our

data points are 2×10 integer valued matrices

$$\begin{bmatrix} a_1 & \cdots & a_{10} \\ b_1 & \cdots & b_{10} \end{bmatrix} \tag{5.1.1}$$

that satisfy Assumption 1.4.2.

Note that classification problems using the weight matrices as features need to be treated as *invariant machine learning* problems. In fact, recall that we have two actions on a weight matrix that keep the corresponding geometric quotient unchanged (so they trivially leave the label terminal/non-terminal unchanged). Explicitly, given a weight matrix as in (5.1.1) there is an action of S_{10} permuting the columns and an action of $\mathrm{GL}_2(\mathbb{Z})$ on the left which reparametrises the action of the torus[14]. Recall that this class of transformations leaves the geometric quotient unchanged [Ahm17]. We approach this group invariant problem by ensuring all our weight matrices are in *standard form*, a specific choice of orbit representative that is consistent for the whole dataset (see [APS23] for a survey on equivariant machine learning and an exposition of fundamental domain projection). In fact, since all $\left(\begin{smallmatrix} a_i \\ b_i \end{smallmatrix}\right)$ lie in a strictly convex cone we can always transform a weight matrix via a $\mathrm{GL}_2(\mathbb{Z})$-transformation into a matrix of the form

$$\begin{bmatrix} a_1 & a_2 & \cdots & a_{10} \\ 0 & b_2 & \cdots & b_{10} \end{bmatrix}, \tag{5.1.2}$$

where the columns are ordered anticlockwise cyclically, $a_{10} < b_{10}$ and $\left(\begin{smallmatrix} a_i \\ b_i \end{smallmatrix}\right) \in \mathbb{Z}^2_{\geq 0}$. We refer to such a matrix as a matrix in *standard form*. Note that any $2 \times N$ weight matrices can be put into standard form, so this can be done for any Picard rank two toric Fano variety for any dimension.

The 2×10 matrices are generated in standard form, as in (5.1.2), where all integer entries are chosen uniformly at random in the range $[0, 7]$, with the following exceptions: $a_1, b_N \in [1, 7]$ and $a_N \in [0, b_N - 1]$. Once a weight matrix has been constructed, we check that it satisfies the conditions in Assumption 1.4.2. We use rejection sampling to create an overall balanced dataset between weight matrices corresponding to terminal and non-terminal varieties. Before generating any weight matrix, a random boolean `True` (terminal) or `False` (non-terminal) is initialised. If the generated matrix satisfies Assumption 1.4.2, the corresponding fan is constructed (as described in Section 1.1.1), otherwise it is discarded. We determine whether the example corresponds to a terminal variety by performing a cone-by-cone analysis of the singularities on the fan (see Section 1.1.2 for details). If the terminality status agrees with the initial boolean, the sample is added to the dataset, otherwise the example is discarded, and the process is repeated until we have a balanced dataset of 5 million terminal examples and 5 million non-terminal examples. We will refer to this dataset as `DSterm_10M`.

Just like in the data generation step for `DSdim_rk2` from Chapter 3, we deduplicate the data using the normal form of the fan (for details see [GK13, KS04]), which we have constructed anyway to check the terminality status of a given weight matrix. We note that even if we are choosing our matrices to be in standard form, there are always two standard

[14]Recall that $\mathrm{GL}_2(\mathbb{Z})$ is the group of 2×2 integer–valued invertible matrices whose inverse is also integer–valued.

forms representative that correspond to the same variety. Therefore, note that we could have deduplicated just by looking at the standard forms directly, which is what we do in Section 5.3.1 (since in that case we are *not* constructing the fan to perform a terminality check, so it would have been way more expensive to construct it just to deduplicate the dataset).

5.2 Neural network

In this section, we build a neural network classifier that determines whether a toric Fano variety (of dimension eight and Picard rank two) is terminal using the flattened weight matrices as features.

5.2.1 Model

All experiments are implemented in Pytorch v.1.13.1 [PGM+19] (the model design, training, and testing) and scikit-learn v1.1.3 [PVG+11] (the data pre-processing step). The code used to perform this analysis is available from Bitbucket [CKV23c] under an MIT licence.

The Multi-Layer Perceptrons (MLP) constructed are feedforward fully-connected neural networks (see Section 2.3.3) trained with binary cross entropy loss function (see Example 2.3.2), leaky ReLU activation (see Example 2.3.1), stochastic mini-batch gradient descent as optimiser. They are trained using early-stopping for a maximum of 100 epochs and learning rate reduction on plateau. Hyperparameter tuning is partly carried out using RayTune [LLN+18] (a popular Python library for hyperparameter tuning at any scale) on a small portion of the training data via random grid search with Async Successive Halving Algorithm (ASHA) scheduler [LJR+20]. Given the best configuration resulting from the random grid search, the most optimal one is chosen by manually exploring the configurations nearby.

5.2.2 Training

The input to our neural network will be a vector of twenty entries given by flattening weight matrices

$$(a_1, \ldots, a_{10}, b_1, \ldots, b_{10}) \in \mathbb{Z}^{20} \, .$$

These vectors are standardised by translating the mean to zero and scaling to variance one (see Section 2.2). Note that this destroys the integer structure, but neural networks are known to perform better on standardised inputs. We train a feedforward fully-connected MLP with three hidden layers on a balanced dataset of terminal and non-terminal varieties to predict terminality. The summary of the network configuration, optimised via hyperparameter tuning, is found in Table 5.1. During training, we keep 20% of the training data as validation, to plot the loss learning curves in Figure 5.1.

Hyperparameter	Value
`Hidden layers`	$(512, 768, 512)$
`Training size`	5 million
`Initial learning rate`	0.01
`Momentum`	0.99
`LeakyReLU slope`	0.01
`Batch size`	128

Table 5.1: Final network architecture and configuration for $\text{MLP}_{5\text{M}}$.

By training using different train–test splits we obtain the training learning curve in Figure 5.1a, which shows that the last train–test split (5 million samples) produces an accurate model that does not overfit. We include all loss learning curves in Figure 5.1. The final accuracy of this MLP is 95% on the remaining testing data, and it produces confusion matrices as in Figure 5.2. We will refer to this neural network as $\text{MLP}_{5\text{M}}$.

The high accuracy of the neural network is very surprising. Despite their fundamental role in algebraic geometry, checking whether a Fano variety has terminal singularities is an extremely challenging question in convex geometry. For the toric case, as we have highlighted in Section 1.1.2, checking if a variety has at worst terminal singularities involves constructing the fan and performing a cone-by-cone analysis of its combinatorics. This is not only expensive from the computational point of view, but it is also unsatisfying from the theoretical one. The result of the neural network suggests the existence of a simpler criterion that uses the weights directly instead of constructing the fan. We explore this direction in Chapter 6 Section 6.1.1.

5.3 Exploring the landscape

The suggestion that a simpler criterion for determining whether a weight matrix correspond to a terminal toric variety is not the only outcome of the high accuracy machine learning model. In fact, we can use the machine learning model to substitute the usual terminality check in order to generate huge amount of data. This would have been practically impossible using the usual computational routines, because they are simply too slow. For example, just generating dataset `DSterm_10M` took 30 CPU years.

5.3.1 Data generation

We want to be able to generate a large amount of terminal toric Fano varieties of Picard rank two, compute their regression data (i.e. A and B from Theorem 3.4.2), and plot them. The aim is to produce plots like Figures 3.18 and 3.20. To do so we implement an AI-assisted data generation workflow that combines algorithmic checks and testing via machine learning, outlined as follows. This dataset is generated using Pytorch v1.13.1 [PGM+19] and SageMath v9.8 [The23].

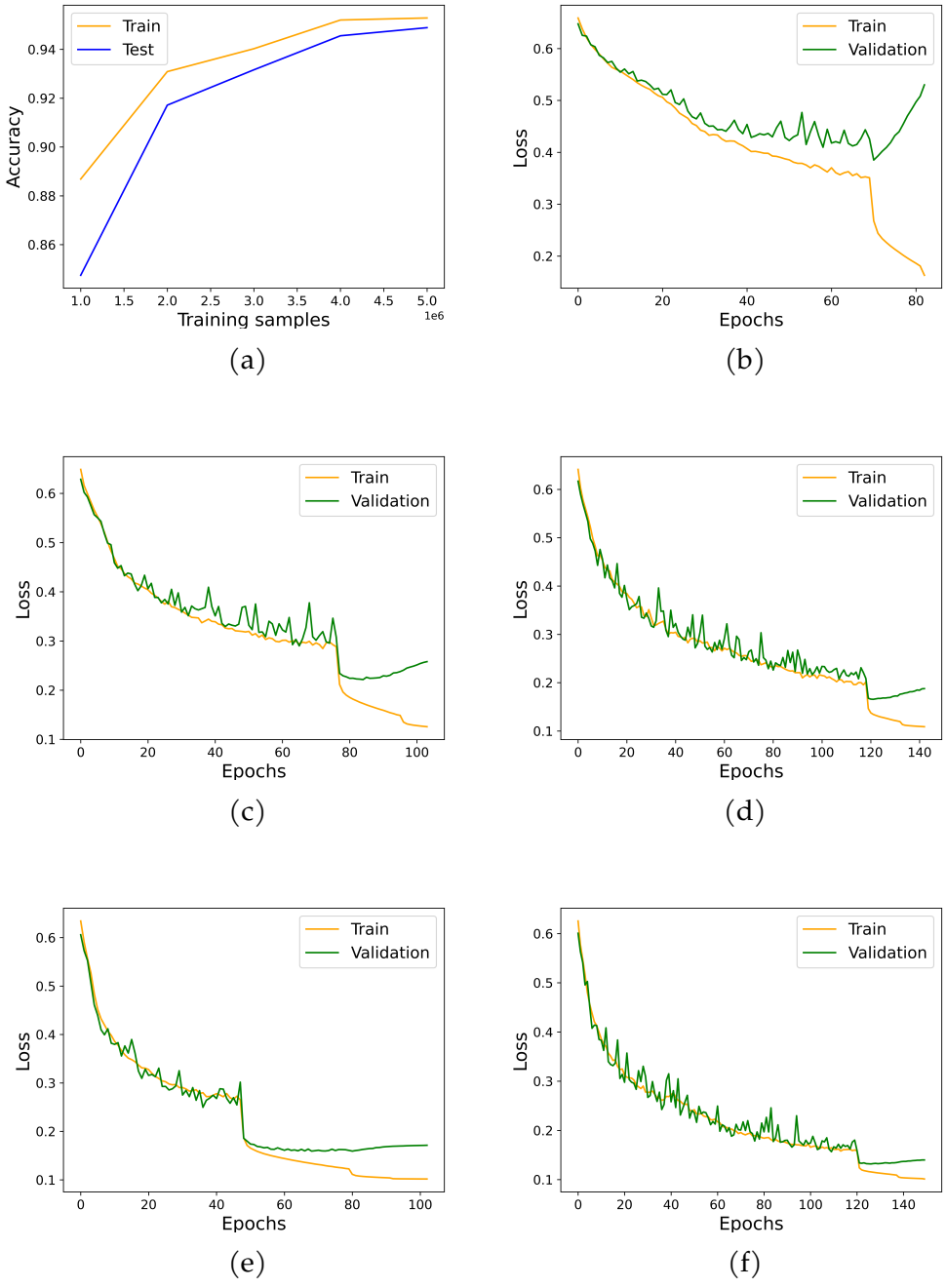
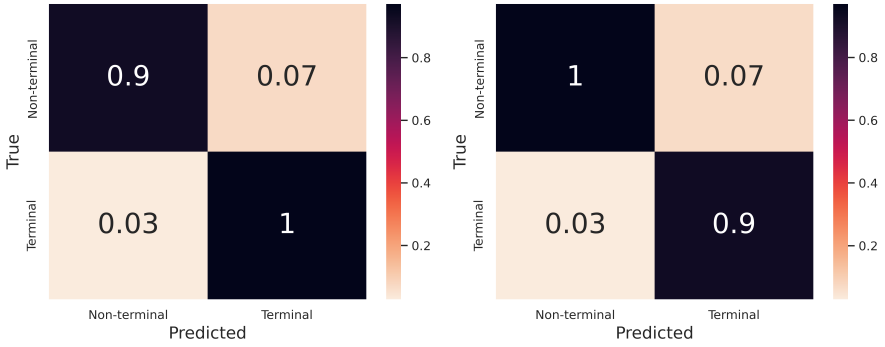


Figure 5.1: Learning curves: (a) number of training samples against accuracy for different train–test splits; (b)–(f) epochs against loss for MLP's trained on $1, 2, 3, 4, 5$ million samples from the dataset `DSterm_10M`



(a) (b)

Figure 5.2: Confusion matrices for $\text{MLP}_{5\text{M}}$: (a) is normalised with respect to the true axis; (b) is normalised with respect to the predicted axis.

1. Generate a random (2×10) integer–valued matrix with entries chosen uniformly at random from in $[0, 7]$.
2. Cyclically order the columns and check if the matrix is in standard form (5.1.2). If it is, we keep it, otherwise go back to Step 1.
3. Check that it satisfy the conditions in Assumption 1.4.2.
4. Check whether the matrix is terminal by the model $\text{MLP}_{5\text{M}}$ constructed in Section 5.2. If the example is classified as terminal we keep it and store its probability, otherwise go back to Step 1.
5. Calculate the regression data (A, B) using formulae in (3.4.3) (here x^* is calculated using the methodology for Picard rank two examples discussed in Section 4.3).
6. If the example is not already in the dataset, insert it, recording the weight matrix, the regression data (A, B), and its Fano index[15].

Remark 5.3.1. Note that in this case, when we are checking whether the sample has already been added to the dataset we are not constructing the normal fan. Instead, we are using the fact that each orbit has at most two representatives in standard form. Therefore, for each standard form weight matrix that we compute, we can compute its other partner in standard form and check if either matrix is already in the dataset.

The final dataset is composed of 100 million samples. A summary of the keys and values of each entry is found in Table 5.2. We call this dataset `DSterm_100M` and note that it contains weight matrices corresponding to *probable* $\mathbb{Q}$-Fano toric Fano varieties of Picard rank two and dimension eight. The *probable* here stands for the fact that the terminality check is carried out with the neural network, hence it is not certain that these varieties have at worst terminal singularities.

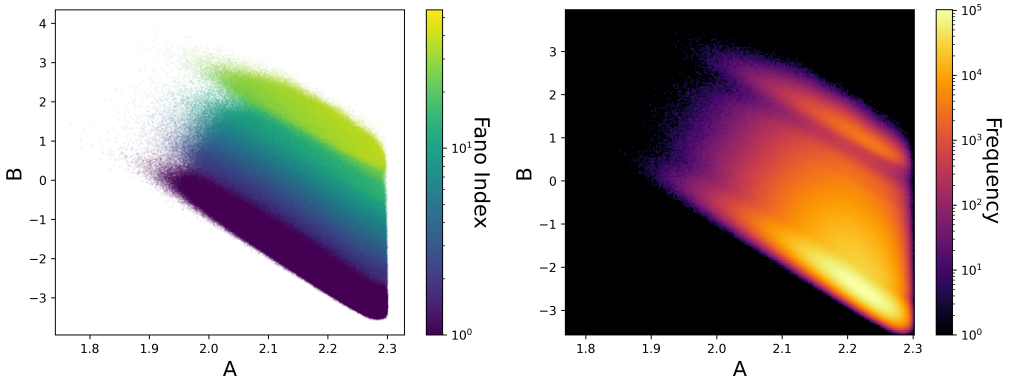
[15]Recall that this is $\gcd(a, b)$ where $a = \sum_{i=1}^{10} a_i$ and $b = \sum_{i=1}^{10} b_i$ for a_i, b_i as in (6.2.2).

Key	Description
`Weights`	Weight matrix, in standard form, given as the two row vectors.
`Regression`	A vector of floats, the regression data (A, B).
`FanoIndex`	An integer, the Fano index.
Total	100 million

Table 5.2: The keys and values for the entries in the dataset `DSterm_100M`.

5.3.2 Data analysis

Given the large amount of data we have generated, we plot A against B, the regression data, for each example; see in Figures 5.3. Note that in the formula for B in (3.4.3), the dependency on the Fano index is logarithmic, and so we have rendered the colour scale logarithmically in Figure 5.3a. In Figure 5.4 we restrict the dataset to low Fano indices. For each Fano index in the range one through to nine, we plot the convex hull of the resulting point cloud. The overlap between these clusters is clear.



(a) (b)

Figure 5.3: The values of A and B from Theorem 3.4.2 for the varieties in dataset `DSterm_100M`. In (a) we colour by Fano index, while in (b) we colour a heatmap according to the frequency. In both cases the colours are rendered logarithmically.

We immediately note that the vertical line that bounds the cluster of Fano varieties is not surprising. In fact, applying the log–sum inequality to the formula of A we obtain that

$$A = -\sum_{i=1}^{N} p_i \log(p_i) \leq -\left(\sum_{i=1}^{N} p_i\right) \log\left(\frac{\sum_{i=1}^{N} p_i}{N}\right) = \log(N)\,.$$

In our case $N = 10$ (since our examples have dimension eight), hence $x = \log(10) \sim 2.3$ is the vertical boundary we see in Figure 5.3. Moreover, we note a linear lower bound for the cluster, and a similar bound was observed and rigorously established for weighted projective spaces Section 3.6.

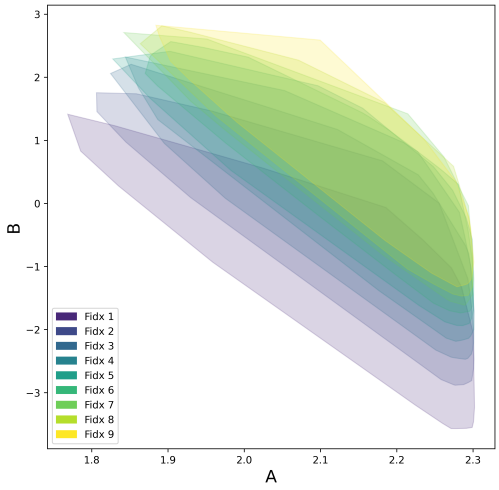
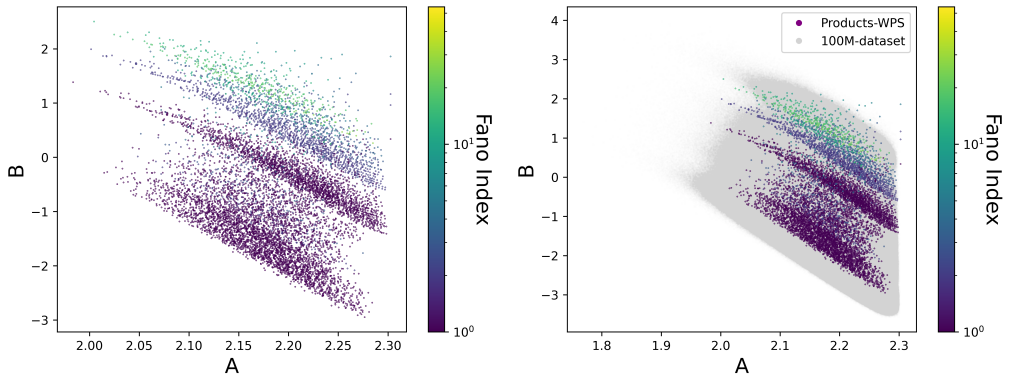


Figure 5.4: Convex hulls obtained from the point clouds of (A, B) for varieties from dataset `DSterm_100M` with Fano indices between one and nine, obtained by projecting to $\mathbb{R}^2$ using the growth coefficients from (3.4.3).

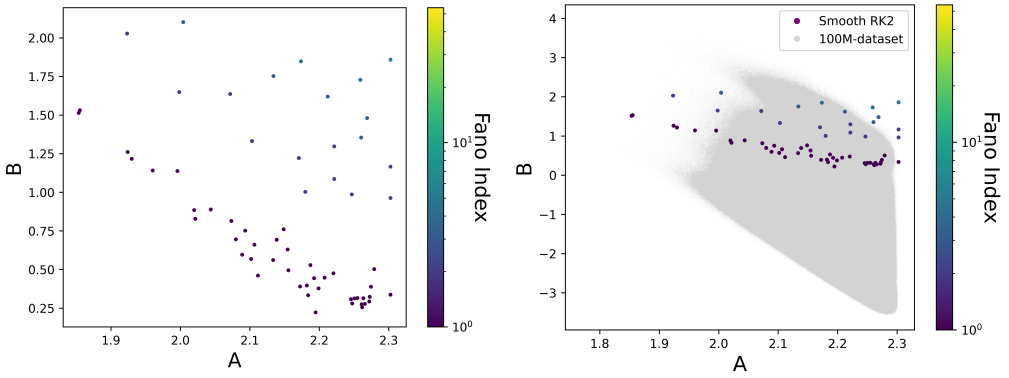


(a) (b)

Figure 5.5: $\mathbb{Q}$-Fano products of weighted projective space in dimension eight, with weights bounded by seven. (a) Projection to $\mathbb{R}^2$ using the growth coefficients from. (b) The same as (a), but plotted on top of (A, B) for varieties in dataset `DSterm_100M`, plotted in grey.

Smooth Fano toric varieties Using the known classification of smooth toric Fano varieties (see Section 1.3) we know that there are 62 smooth Fano toric varieties in dimension eight and of Picard rank two, all of which have weights bounded by seven when expressed in standard form (5.1.2). These are plotted in Figure 5.6, and appear to fall in the upper extreme region within each cluster. This could suggest that one of the directions of the cluster is recording the complexity of the singularities.

Products of weighted projective space To better understand this clustering by Fano index, we consider the simplest $\mathbb{Q}$-factorial Fano toric varieties of Picard rank two: products of weighted projective spaces. Recall from Section 1.1 that a product of weighted projective



(a) (b)

Figure 5.6: The smooth Fano toric varieties in dimension eight and of Picard rank two. (a) Projection to $\mathbb{R}^2$ using the growth coefficients from (3.4.3). (b) The same as (a), but plotted on top of (A, B) for varieties in dataset `DSterm_100M`, plotted in grey.

spaces $X = \mathbb{P}(a_0, \ldots, a_N)$ and $Y = \mathbb{P}(b_0, \ldots, b_M)$ is specified by a weight matrix

$$\begin{bmatrix} a_0 & \cdots & a_N & 0 & \cdots & 0 \\ 0 & \cdots & 0 & b_0 & \cdots & b_M \end{bmatrix}.$$

This matrix determines a $\mathbb{Q}$-factorial Fano toric variety of Picard rank two and dimension $N + M$, denoted $X \times Y$. The singular points of $X \times Y$ are determined by the singular points of X and Y. In particular, $X \times Y$ is terminal if and only if both X and Y are terminal. Recall, that we can test terminality for weighted projective spaces using Proposition 1.1.3.

We can then enumerate all terminal weighted projective spaces in dimensions one to seven, with weights $1 \leq a_i \leq 7$. The number in each dimension is given in Table 5.3. By taking products, we obtain 8792 distinct $\mathbb{Q}$-Fano toric varieties of Picard rank two in dimension eight; these examples are plotted in Figure 5.5. This supports our observation that the $\mathbb{Q}$-Fano varieties fall into large overlapping clusters that are determined by the Fano index. Note that the products of weighted projective space appear to fall within the upper region of each cluster.

d	1	2	3	4	5	6	7
#	1	1	7	80	356	972	2088

Table 5.3: The number of terminal weighted projective spaces in dimension d, $1 \leq d \leq 7$, with weights a_i bounded by seven.

A cluster of high-Fano index examples Curiously, Figure 5.3b shows a cluster of high-Fano index cases (at the top of the plot) standing apart from the remainder of the data. This is also visible in Figure 5.7 which shows the frequency distribution of the Fano indix across the dataset. This can be explained as follows. The uptick in frequencies in the histogram in Figure 5.7 can be justified by the following argument. Consider how many ways we can write n as a sum of ten numbers between 0 and 7 (inclusive, and with possible repeats). This resembles a normal distribution with $n = 35$: the integer with most

different ways of writing it as a sum. Therefore, the higher proportion of these high Fano index examples is due to our sampling constraints on the entries of the weight matrix: amongst those matrices that have $a = b$ we have the highest probability of selecting one that has $a = b = 35$. Therefore, we see a misleading accumulation around those Fano indices: if we were sampling with a bigger weight bound we would see this cluster higher up.

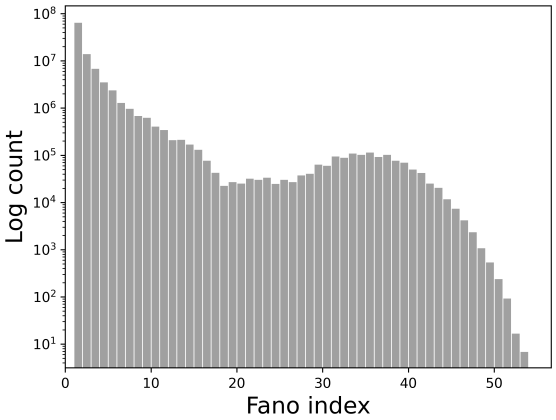
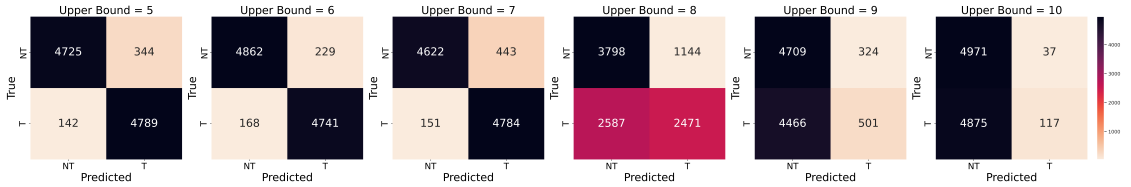


Figure 5.7: Distribution of the Fano index $\gcd(a, b)$ for the varieties in dataset `DSterm_100M` (note that the vertical axis scale is logged).

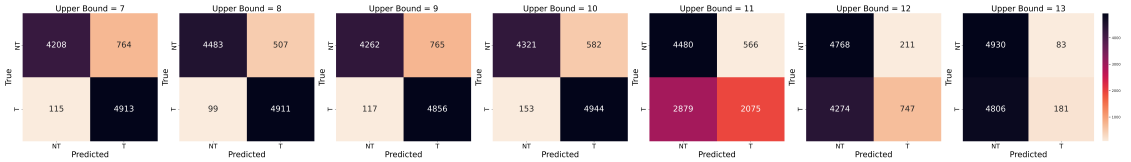
5.4 Limitations and outlook

Out-of-sample performance Despite its potential, this approach has some limitations: tackling these problems will guide further research. A key problem with our workflow is that our classifier performs poorly on out-of-sample data. In fact, our weight matrices are generated with weights bounded by seven. To test the performance out-of-sample we generate $10\,000$ random weight matrices with weight bound increasing from five to ten, balanced between terminal and non-terminal. The data generation steps for these samples are the same as the data generation steps for `DSterm_10M`. For those weight matrices with weights bounded by seven (or smaller bounds) the model is extremely accurate (95%). However, when we increase the bound the accuracy quickly drops: 62% for entries bounded by eight; 52% for entries bounded by nine; and 50% for entries bounded by ten. These results are summarised in the confusion matrices in Figure 5.8a, which highlight how the network quickly degenerated to always predicting non-terminal example.

Because of this, it is natural to ask what kind of statement the neural network is approximating. Is it actually picking up some genuine mathematical structure or is it just the result of some finite-size effect due to the choice of bound on the weights (which is seven in our case)? Intuitively, we expect the neural network to be approximating an actual mathematical statement, given that the training and testing data are free of noise and the final accuracy is so high. However, the poor out-of-sample performance indicates that we might not know which kind of mathematical statement the network is approximating. There are two possibilities.



(a) $\text{MLP}_{5\text{M}}$, trained on `DSterm_10M`, which contains weight matrices with bound on the entries seven.



(b) $\text{MLP}_{10\text{M}}$, trained on `DSterm_20M`, which contains weight matrices with bound on the entries ten.

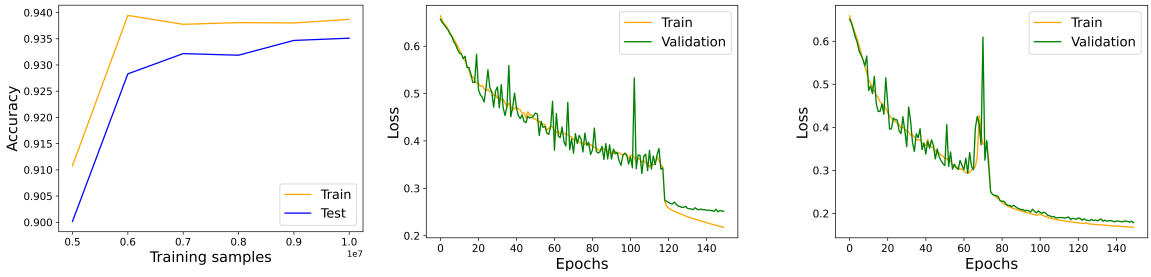
Figure 5.8: Confusion matrices for the neural network classifier on in-sample and out-of-sample data. In each case a balanced set of $10\,000$ random examples was tested. Note that the confusion matrices are not normalised with respect to neither the true nor the predicted axis.

- The neural network might be approximating a general statement for detecting terminality of Picard rank two toric Fano varieties. In this case, the poor out-of-sample performance could be because the network is approximating such a statement in a way that does not generalise well to matrices with bigger weights. This is a common phenomenon when using neural networks.
- The neural network might be approximating a statement that only applies to weight matrices with small entries, which mathematically correspond to Fano varieties with terminal singularities of small index[16]. If this is true we would expect not to be able to build a corresponding high accuracy network for weight matrices with bigger bounds on the weights. In this case, the drop in accuracy for higher bound is justified by the fact that the actual mathematical statement that is being approximate requires a small bound on the weights.

To understand this we perform the same experiment for weight matrices that have weights bounded by ten, instead of seven. Using the same data generation workflow as in Section 5.1 we generate a new dataset of 20 million balanced examples labelled terminal and non-terminal. The data generation steps are exactly the same except for the terminality check, for which we were now able to use the new algorithm explained in Chapter 6, whose formulation came after the success of the neural network $\text{MLP}_{5\text{M}}$. The new algorithm is roughly fifteen times faster than the original method, and therefore this allowed us to comfortably generate more data compared to `DSterm_10M`. We call this new dataset `DSterm_20M`.

We train a fully-connected feed-forward neural network with the same architecture as the original neural network $\text{MLP}_{5\text{M}}$; recall the final configuration from Table 5.1. The network was trained on the flattened weight matrices, where we applied a standard scaler to each entry. It was trained using binary cross-entropy as loss function, stochastic mini-batch

[16]The index of a singularity $P \in X$ is the smallest r such that rK_X is Cartier in a neighbourhood of P.



(a) (b) (c)

Figure 5.9: (a) Accuracy for different train–test splits; (b) epochs against loss for the network trained on 5 million samples of `DSterm_20M`; (c) epochs against loss for the network trained on 10 million samples of `DSterm_20M`.

gradient descent optimiser and using early-stopping, for a maximum of 150 epochs and with learning rate reduction on plateaux. Using 5 million balanced samples for training (80% for training and 20% for validation) and testing on 15 million samples, the network obtained an accuracy of 90% – Figure 5.9b displays the loss learning curve. We were able to recover a 94% accuracy by training on 10 million samples – Figure 5.9c displays the loss learning curve for this case. Moreover, we record the training and validation accuracies for intermediate training sizes in Figure 5.9a.

Therefore, training on dataset `DSterm_20M` was also able to produce a high testing accuracy, as with the original bounds (seven) on the weights. However, it is important to note that this made possible only by using a much larger training sample, which is a testament to the increased difficulty of the task. In addition to this, the plateau shown in the learning curve in Figure 5.9a suggests that increasing the training set size further will not lead to an increase in accuracy, meaning that we might have reached the limits of this model.

Overall, the fact that we were able to build a high-accuracy neural network for a larger weight bound supports the argument that the model is actually approximating a general mathematical statement, but in a way that does not generalise well to higher bound. Nonetheless, the new chosen bound (ten) is still too low to exclude the possibility that the model is actually capturing a mathematical statement that needs weight matrices with small entries. To reach a definite conclusion about this, similar studies with even larger bounds should be carried out (and they could potentially suggest if the cut-off bound of the weight exists, and if it does what it is).

Training size Moreover, an additional limitation of this approach is that the training process appears to need more samples that it would be ideal, since the accurate computational check that a toric variety is terminal is incredibly time-consuming (especially if we were to consider examples of dimension much higher than eight). We can imagine that a more sophisticated model (potentially able to capture the group-invariance nature of the problem) might require less training data that the chosen architecture.

However, we also remark that the design of the new terminality algorithm for Picard rank

two varieties (which we describe and prove in Chapter 6) is fifteen times faster than the naïve cone-by-cone analysis described in Chapter 1. Therefore, adopting this new algorithm greatly improves the data generation workflow, making repeating this analysis for other dimension (and/or weight bounds) much more accessible.

Beyond toric varieties Lastly, we have only been working with toric varieties of Picard rank two, which are a special class of $\mathbb{Q}$-Fano varieties. Higher rank toric varieties are definitely accessible by similar tools, however their vectorisation might require a more sophisticated study of tackling the groups acting on the space of weight matrices. In fact, in the construction of an MLP for the Picard rank two case we rely on the existence of a standard form (5.1.2) – for weight matrices, which allows us to pick an almost unique representative of each group orbit. It is not clear whether there exists an equally nice standard form for weight matrices in higher rank, therefore a possibility would be to tackle the group action by building a machine learning invariant model (either with a preprocessing step, data augmentation, or by choosing the correct architecture).

Dealing with general $\mathbb{Q}$-Fano varieties seems unapproachable, however if we are only interested in classifying algebraic varieties up to deformation, the situation is better than expected. In fact, where we know the classification (see Table 1.1), any smooth Fano variety in low dimension is either (up to deformation) a toric variety, a toric complete intersection, or a quiver flag zero locus [CCGK16, Kal19]. The hope here is that the (currently unknown) classification of $\mathbb{Q}$-Fano varieties will share this property of mostly being covered by varieties of these types. This is especially important since each of these classes of varieties has a geometry that is highly controlled by combinatorics (just like toric varieties). This sheds some hope that their combinatorial vectorisation makes them amenable to similar data-driven tools.

The best possible path forward would be to train an explainable model that predicted terminality from the weight data. This would allow us to extract from the machine learning not only that the problem is tractable, but also a precise mathematical conjecture for the solution. At the moment, however, we are very far from this. The multilayer perceptron that we trained is a black-box model, and post-hoc explanatory methods such as SHAP analysis [LL17] yielded little insight: all features were used uniformly, as might be expected; see Figure 5.10. Of course, b_1 has no influence, since it is always zero, but apart from this nothing stands out as particularly influential. Interestingly, a low value of a feature (our lowest value is zero) is very influential for the outcome of the model for many – but not all – the features.

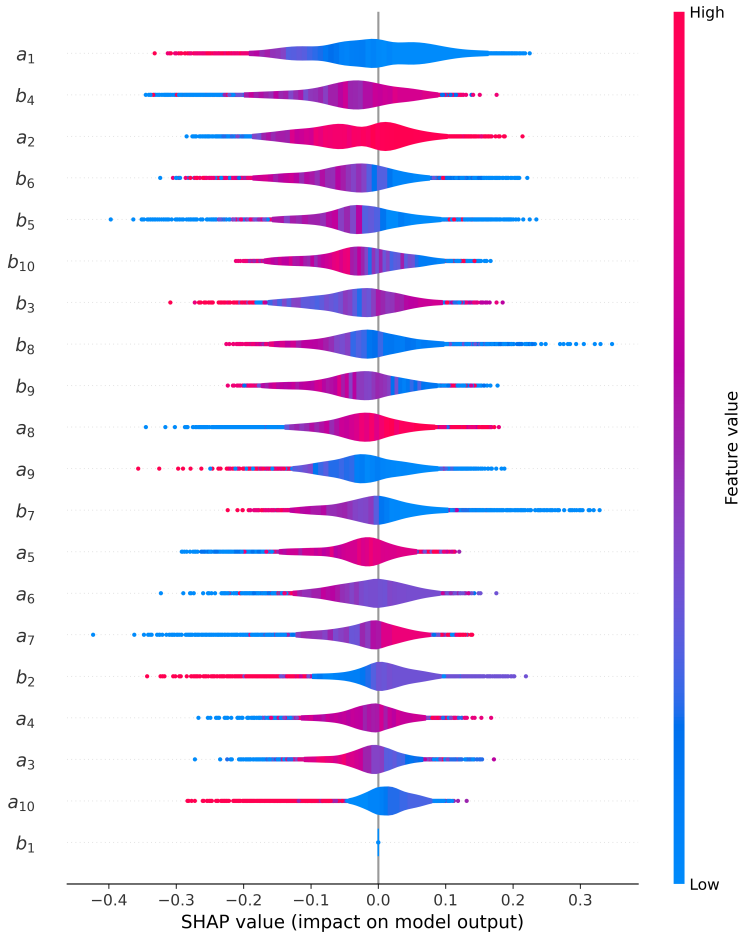


Figure 5.10: Model sensitivity analysis using SHAP values, for the model $\text{MLP}_{5\text{M}}$ trained on the data coming from `DSterm_10M`. It predicts terminality with 95% accuracy.

6 Algorithms for Toric Fano Varieties

In this chapter, we collect results related to the construction of algorithms for toric Fano varieties. The first part develops a new algorithm to test for terminal singularities for toric Fano varieties of Picard rank two. This is originally from [CKV24] where it was designed after the construction of the neural network that detects terminal singularities which we have presented in Chapter 5. We then use the same methodology to generalise to a new algorithm to check if a toric Fano varieties of Picard rank two has at worst canonical singularities. We conclude this part by discussing why the methodology does not extend to Picard rank higher than two in a straightforward way.

In the second part of the chapter, we sketch some work in progress that aims to answer the following question

"Does a sequence $(p_i)_{i=0}^n$ arise as the period sequence of a toric Fano variety?"

This is current ongoing work, and we do not answer this question in full by the end of the chapter. However, we sketch some initial progress and produce an algorithm that answers this question in the case of weighted projective spaces and a special case of Picard rank two varieties.

We include pseudocode for the algorithms mentioned and the implementation of the algorithms is in Python and SageMath [The23] and is found at [Ven24b].

6.1 Terminal and canonical singularities

In this section we describe a new global criterion for determining whether a $\mathbb{Q}$-factorial toric Fano variety of Picard rank two has at worst terminal singularities. This result is inspired by the high accuracy neural network model from Chapter 5 (but unlike the machine learning model, this is an exact method that is not restricted just to dimension eight). Then, we extend this result to checking whether a $\mathbb{Q}$-factorial toric Fano variety of Picard rank two has at worst canonical singularities. Finally, we discuss why it is difficult to extend this methodology to varieties with Picard rank greater than two.

6.1.1 Picard rank two

Terminal singularities We prove a theoretical result, Proposition 6.1.3, which leads to a new algorithm for checking terminality directly from the weight matrix, for $\mathbb{Q}$-factorial toric Fano varieties of Picard rank two (these steps are made explicit in Algorithm 2). Consider a weight matrix as in (1.1.2),

$$\begin{bmatrix} a_1 & \cdots & a_N \\ b_1 & \cdots & b_N \end{bmatrix}$$

satisfying the conditions in Assumption 1.4.2. Let X be the toric Fano variety that it specifies. As discussed in Section 1.1.1 we can build a convex polytope P corresponding to the weight matrix. The polytope $P \subset \mathbb{R}^{N-2}$ has N vertices given by the primitive lattice points on the N rays of the corresponding fan, $\{e_1, \ldots, e_N\}$. They satisfy the following system

$$\begin{cases} a_1 e_1 + \cdots + a_N e_N = 0 \\ b_1 e_1 + \cdots + b_N e_N = 0 \end{cases}.$$

Then, X has terminal singularities if and only if the only lattice points in P are the vertices $\{e_1, \ldots, e_N\}$ and the origin; see Section 1.1.2 for a detailed discussion of this.

Definition 6.1.1. Let Δ_i denote the convex hull in $\mathbb{R}^{N-2}$ with vertices $\{e_1, \ldots, \hat{e}_i, \ldots, e_N\}$. We say that Δ_i is *mostly empty* if each lattice point in Δ_i is either a vertex or the origin.

Remark 6.1.2. Note that in our setup (where X is $\mathbb{Q}$-factorial and has Picard rank two) Δ_i will be a simplex for all $i \in \{1, \ldots, N\}$.

Proposition 6.1.3. *Consider an integer–valued weight matrix*

$$\begin{bmatrix} a_1 & \cdots & a_N \\ b_1 & \cdots & b_N \end{bmatrix}$$

that satisfies Assumption 1.4.2. Let $g_i = \gcd(a_i, b_i)$, and let $A_i, B_i \in \mathbb{Z}$ such that $A_i a_i + B_i b_i = g_i$. Set

$$\alpha_i^j = \frac{a_j b_i - b_j a_i}{g_i} \qquad \alpha_i = \sum_{j=1}^{N} \alpha_i^j$$

$$\beta_i^j = -A_i a_j - B_i b_j \qquad \beta_i = \sum_{j=1}^{N} \beta_i^j \qquad f_i = \frac{\alpha_i g_i}{\gcd(g_i, \beta_i)}$$

noting that all these quantities are integers. Then Δ_i is mostly empty if and only if for all $k \in \{0, \ldots, f_i - 1\}$ and $l \in \{0, \ldots, g_i - 1\}$ such that

$$\sum_{j \neq i} \left\{ k \frac{\alpha_i^j}{f_i} + l \frac{\beta_i^j}{g_i} \right\} = 1$$

we have that

$$\left\{ k \frac{\alpha_i^j}{f_i} + l \frac{\beta_i^j}{g_i} \right\} = \frac{\alpha_i^j}{\alpha_i}$$

for all $j \neq i$. Here $\{q\} = q - \lfloor q \rfloor$ for $q \in \mathbb{Q}$.

Proof. Recall that we can construct a convex polytope P from the given weight matrix with vertices $e_1, \ldots, e_N \in \mathbb{Z}^{N-2}$, and that

$$\begin{cases} a_1 e_1 + \cdots + a_N e_N &= 0 \\ b_1 e_1 + \cdots + b_N e_N &= 0 \end{cases}$$

where a_i's and b_j's are the entries in the weight matrix. The same argument applied to the equivalent weight matrix

$$\begin{bmatrix} b_i/g_i & -a_i/g_i \\ A_i & B_i \end{bmatrix} \cdot \begin{bmatrix} a_1 & \cdots & a_N \\ b_1 & \cdots & b_N \end{bmatrix}$$

gives barycentric co-ordinates for the origin and e_i in terms of the remaining vertices of Δ_i,

$$\begin{cases} \alpha_i^1 e_1 + \cdots + \alpha_i^{i-1} e_{i-1} + \alpha_i^{i+1} e_{i+1} + \cdots + \alpha_i^N e_N &= 0 \\ \beta_i^1 e_1 + \cdots + \beta_i^{i-1} e_{i-1} + \beta_i^{i+1} e_{i+1} + \cdots + \beta_i^N e_N &= g_i e_i \end{cases}.$$

Fix $i \in \{1, 2, \ldots, N\}$. Define $u \colon \mathbb{Q}^{N-1} \to \mathbb{Q}$ by $u(x_1, \ldots, x_{N-1}) = x_1 + \cdots + x_{N-1}$, and let Ψ denote the lattice

$$\{v \in \mathcal{Z} \mid u(v) = 1\}$$

where $\mathcal{Z}$ is the span over $\mathbb{Z}$ of the standard basis $E_1, \ldots, E_{N-1}$ for $\mathbb{Q}^{N-1}$ together with

$$\frac{1}{f_i}(\alpha_1^1, \ldots, \hat{\alpha}_i^i, \ldots, \alpha_1^N) \quad \text{and} \quad \frac{1}{g_i}(\beta_1^1, \ldots, \hat{\beta}_i^i, \ldots, \beta_1^N)\,.$$

Here the $\hat{}$ indicates that the ith entry in each vector is omitted. We define $\phi : \Psi \to \mathbb{Z}^{N-2}$ to be the $\mathbb{Z}$-linear map that sends $E_1, \ldots, E_{N-1}$ to $e_1, \ldots, \hat{e}_i, \ldots, e_N$ and

$$\phi\left(\frac{1}{f_i}(\alpha_i^1, \ldots, \hat{\alpha}_i^i, \ldots, \alpha_i^N)\right) = 0\,, \quad \phi\left(\frac{1}{g_i}(\beta_i^1, \ldots, \hat{\beta}_i^i, \ldots, \beta_i^N)\right) = e_i\,.$$

It is easy to see that ϕ is well-defined and bijective.

Consider the higher-dimensional parallelepiped Γ in $\mathcal{Z}$ generated by the standard basis of $\mathbb{Z}^{N-1}$. We note that each lattice point of $\mathcal{Z}$ in Γ can be represented as a linear combination

$$\frac{k}{f_i}(\alpha_i^1, \ldots, \hat{\alpha}_i^i, \ldots, \alpha_i^N) + \frac{l}{g_i}(\beta_i^1, \ldots, \hat{\beta}_i^i, \ldots, \beta_i^N) \tag{6.1.1}$$

for some $k \in \{0, \ldots, f_i - 1\}$ and $l \in \{0, \ldots, g_i - 1\}$; this representation is unique if and only if the vertices of Δ_i $\mathbb{Z}$-span $\mathbb{Z}^{N-2}$. Hence, Δ_i is mostly empty if and only if whenever

$$\sum_{j \neq i} \left\{ k\frac{\alpha_i^j}{f_i} + l\frac{\beta_i^j}{g_i} \right\} = 1 \tag{6.1.2}$$

we have that the linear combination in (6.1.1) represents the origin. However, this is the

case if and only if

$$\left\{ k\frac{\alpha_i^j}{f_i} + l\frac{\beta_i^j}{g_i} \right\} = \frac{\alpha_i^j}{\alpha_i}$$

for all j, since $(k, l) = (\frac{f_i}{\alpha_i}, 0)$ represents the origin by construction. □

Let $s_+ = \{i \mid a_i b - b_i a > 0\}$, $s_- = \{i \mid a_i b - b_i a < 0\}$, and let I be the smallest of s_+ and s_-. Then $\{\Delta_i \mid i \in I\}$ forms a triangulation of P by the following lemma.

Lemma 6.1.4. *Given an integer–valued weight matrix*

$$\begin{bmatrix} a_1 & \cdots & a_N \\ b_1 & \cdots & b_N \end{bmatrix}$$

that satisfies Assumption 1.4.2, let

$$s_+ = \{i \mid a_i b - b_i a > 0\}$$
$$s_- = \{i \mid a_i b - b_i a < 0\}$$

for $a = \sum_{i=1}^N a_i$ and $b = \sum_{i=1}^N b_i$. Then, $\{\Delta_i \mid i \in s_+\}$ and $\{\Delta_i \mid i \in s_-\}$ both form a triangulation of P by $(N-2)$-dimensional simplices, where P is a spanning polytope obtained from the weight matrix.

Proof. Recall that Δ_i is defined as the convex hull of $\{e_1, \ldots, \hat{e}_i, \ldots, e_N\}$, where $\{e_1, \ldots, e_N\}$ are the vertices of P. Then Δ_i and Δ_j belong to a triangulation into simplices of P if and only if

$$\Delta_i \cap \Delta_j = \text{Conv}\{e_k \mid k \neq i, j\} \,.$$

The inclusion of the convex hull into the intersection is clear, and equality holds if and only if the cone

$$\sigma = \text{Cone}\{e_k \mid k \neq i, j\}$$

is not a cone in Σ (the fan corresponding to the polytope P) given the stability condition $\omega = \binom{a}{b}$. Recall from Section 1.1.1 that $\sigma \notin \Sigma$ if and only if $\{i, j\} \notin \mathcal{A}_\omega$. This holds if and only if $\binom{a_i}{b_i}$ and $\binom{a_i}{b_i}$ are on the same side of the stability condition, i.e.

$$(a_i b - a b_i)(a_j b - a b_j) > 0 \,,$$

i.e. $i, j \in s_-$ or s_+. □

Thus, a $\mathbb{Q}$-factorial toric Fano variety with Picard rank two has terminal singularities if and only if Δ_i is mostly empty for each $i \in I$. We outline these steps in Algorithm 2.

Remark 6.1.5. Testing on $100\,000$ randomly-chosen examples indicates that Algorithm 2 is approximately 15 times faster than the fan-based approach to checking terminality (0.020s per weight matrix for Algorithm 2 versus 0.305s for the standard approach, both implemented in Magma). On single examples, the neural network classifier from Chapter 5 (MLP_{5M}) is approximately 30 times faster than Algorithm 2. The neural network also benefits greatly from batching, whereas the other two algorithms do not: for batches of size $10\,000$, the neural network is roughly 2000 times faster than Algorithm 2.

Algorithm 2: The algorithm checks if a $\mathbb{Q}$-factorial toric Fano variety of Picard rank two (given by a weight matrix $2 \times N$) has at worst terminal singularities.

```
Function IsTerminal(((a_1, ..., a_N), (b_1, ..., b_N)): list of lists): is
    a ←∑_{i=1}^N a_i, b ←∑_{i=1}^N b_i;
    s_+ ←{i | a_i b − b_i a > 0}, s_− ←{i | a_i b − b_i a < 0};
    I ←the smallest of s_+ and s_−;
    for i in I do
        Test if Δ_i is mostly empty, using Proposition 6.1.3.;
        if Δ_i is not mostly empty then
            Return False
        end
    end
    Return True;
end
```

Canonical singularities Canonical singularities[17] are a less restrictive class of singularities than terminal singularities: if an algebraic variety has at worst canonical singularities then it has at worst terminal singularities. As for terminal singularities, they have a nice combinatorial characterisation in the case of toric Fano varieties. In fact, given a toric Fano variety X of dimension D and its associated spanning polytope $P \subset \mathbb{R}^D$, then X has at worst canonical singularities if and only if

$$P^\circ \cap \mathbb{Z}^D = \{\mathbf{0}\} ,$$

where P° is the interior of P.

Therefore, we can easily generalise our above result to a test for canonical singularities in the case of Picard rank two $\mathbb{Q}$-factorial toric Fano varieties. This is achieved by the same mechanisms as Algorithm 2 to check that $P^\circ \cap \mathbb{Z}^D = \{\mathbf{0}\}$ rather than $P \cap \mathbb{Z}^D = \text{vert}(P) \cup \{\mathbf{0}\}$ (in our case of Picard rank two we will have, as usual, $D = N - 2$, where N is the number of columns in the weight matrix).

Let us define an equivalent to Definition 6.1.1 property for this case.

Definition 6.1.6. Let Δ_i denote the convex hull in $\mathbb{R}^{N-2}$ with vertices $\{e_1, \ldots, \hat{e}_i, \ldots, e_N\}$. We say that $\Delta_i \setminus \partial P$ is *mostly empty* if $\Delta_i \setminus (\partial\Delta_i \cap \partial P)$ is either empty or just contains the origin.

We can then generalise Proposition 6.1.3 to check if $\Delta_i \setminus \partial P$ is mostly empty.

Proposition 6.1.7. *Consider a weight matrix*

$$\begin{bmatrix} a_1 & \cdots & a_N \\ b_1 & \cdots & b_N \end{bmatrix}$$

[17] A variety X has *canonical singularities* if it satisfies two conditions: there exists $r \in \mathbb{Z}_{\geq 1}$ such that rK_X is Cartier; if $f : X \to Y$ is a resolution of singularities and $\{E_i\}_i$ is the family of all exceptional prime divisors of f then $rK_y = f^*(rK_X) + \sum_i a_i E_i$ for $a_i \geq 0$; see [Rei87].

that satisfies Assumption 1.4.2. Let $g_i = \gcd(a_i, b_i)$*, and let* $A_i, B_i \in \mathbb{Z}$ *such that* $A_i a_i + B_i b_i = g_i$*. Set*

$$\alpha_i^j = \frac{a_j b_i - b_j a_i}{g_i} \qquad \alpha_i = \sum_{j=1}^{N} \alpha_i^j$$

$$\beta_i^j = -A_i a_j - B_i b_j \qquad \beta_i = \sum_{j=1}^{N} \beta_i^j \qquad f_i = \frac{\alpha_i g_i}{\gcd(g_i, \beta_i)}$$

noting that all these quantities are integers. Define Ω *as the set of those vectors*

$$\left(\left\{ k\frac{\alpha_i^1}{f_i} + l\frac{\beta_i^1}{g_i} \right\}, \ldots, \widehat{\left\{ k\frac{\alpha_i^i}{f_i} + l\frac{\beta_i^i}{g_i} \right\}} \ldots, \left\{ k\frac{\alpha_i^N}{f_i} + l\frac{\beta_i^N}{g_i} \right\} \right)$$

for $k \in \{0, \ldots, f_i - 1\}$ *and* $l \in \{0, \ldots, g_i - 1\}$*, such that*

$$\sum_{j \neq i} \left\{ k\frac{\alpha_i^j}{f_i} + l\frac{\beta_i^j}{g_i} \right\} = 1$$

we have that

$$\left\{ k\frac{\alpha_i^j}{f_i} + l\frac{\beta_i^j}{g_i} \right\} \neq \frac{\alpha_i^j}{\alpha_i}$$

for some j*. The* $\Delta_i \setminus \partial P$ *is mostly empty if and only if either* $\Omega = \varnothing$ *or* $\forall v \in \Omega$*, there exists* $j \in \{j \mid v_j = 0\}$ *such that*

$$(ab_{j+\mathbb{1}_{j \geq i}} - a_{j+\mathbb{1}_{j \geq i}} b)(ab_i - a_i b) < 0\,,$$

where $a = \sum_{i=1}^{N} a_i$ *and* $b = \sum_{i=1}^{N} b_i$ *and* $k = j + \mathbb{1}_{j \geq i}$*.*

Proof. The construction of the set Ω is analogous to Proposition 6.1.3. There, we had defined a surjection from the lattice points in Δ_i to the lattice point of $\mathcal{Z}$ in the convex hull of the standard basis vectors $E_1, \ldots, E_{N-1}$ in $\mathbb{Q}^{N-1}$, where $\mathcal{Z}$ is the span over $\mathbb{Z}$ of the standard basis $E_1, \ldots, E_{N-1}$ for $\mathbb{Q}^{N-1}$ together with

$$\frac{1}{f_i}(\alpha_i^1, \ldots, \hat{\alpha}_i^i, \ldots, \alpha_i^N) \qquad \text{and} \qquad \frac{1}{g_i}(\beta_i^1, \ldots, \hat{\beta}_i^i, \ldots, \beta_i^N)$$

Then, Ω is the set of these lattice points, unless they correspond to the origin (which might or might not be inside Δ_i). Therefore, $\Omega = \varnothing$ is equivalent to Δ_i being mostly empty, the case for Proposition 6.1.3. If Δ_i is not mostly empty, but $\Delta_i \setminus \partial P$ is mostly empty, then $\varnothing \neq \Omega \subset \partial\Delta_i \cap \partial P$. Therefore, this is equivalent to for all $v \in \Omega$ to lie in a facet of the convex hull of the standard basis vectors in $\mathbb{Q}^{N-1}$, corresponding to a facet of P. We can check whether the convex hull of a subset of vertices is a facet of the polytope by checking whether it generates a cone in the fan, so combinatorially this translates to the condition that there exists $j \in \{j \mid v_j = 0\}$ such that the $(j + \mathbb{1}_{j \geq i})$th and the ith columns of the weight matrix lie on different sides of the stability condition, i.e.

$$(ab_{j+\mathbb{1}_{j \geq i}} - a_{j+\mathbb{1}_{j \geq i}} b)(ab_i - a_i b) < 0\,,$$

where $a = \sum_{i=1}^{N} a_i$ and $b = \sum_{i=1}^{N} b_i$. Note that the shift in index is because $v \in \Omega$ is such that $v = \sum_{j=1}^{N-1} v_j E_j$ and $\Phi(v) = \sum_{j=1}^{i-1} v_j e_j + \sum_{j=i+1}^{N} v_j e_j$ where Φ is the bijection as in the proof of Proposition 6.1.3. □

As for the terminal case, we outline the steps for checking if a $\mathbb{Q}$-factorial toric Fano variety with Picard rank two has at worst canonical singularities using this proposition in Algorithm 3, using the fact that we can triangulate P into simplices using Lemma 6.1.4.

Algorithm 3: The algorithm checks if a $\mathbb{Q}$-factorial toric Fano variety of Picard rank two (given by a weight matrix $2 \times N$) has at worst canonical singularities.

```
Function IsCanonical(((a_1, ..., a_N), (b_1, ..., b_N)): list of lists): is
    a ←∑_{i=1}^{N} a_i, b ←∑_{i=1}^{N} b_i;
    s_+ ←{i | a_i b − b_i a > 0}, s_− ←{i | a_i b − b_i a < 0};
    I ←the smallest of s_+ and s_−;
    for i in I do
        Test if Δ_i \ ∂P is mostly empty, using Proposition 6.1.7.;
        if Δ_i \ ∂P is not mostly empty then
            Return False
        end
    end
    Return True;
end
```

6.1.2 Higher Picard rank

We note that similar ideas seem to be applicable Picard rank higher than two. Consider a $\mathbb{Q}$-factorial toric Fano variety X of Picard rank r and dimension D. Its geometrical information is encoded in an $r \times N$ integer valued matrix with columns $(\alpha_i)_i$ – as seen in Chapter 1 where $N = D + r$. Such a variety will correspond to a lattice polytope P in $\mathbb{Z}^D$: one might imagine triangulating such a polytope into D-dimensional simplices and applying a criterion similar to Proposition 6.1.3 (or Proposition 6.1.7 in the canonical case) to check if each simplex is mostly empty. However, the higher Picard case needs more careful thoughts.

In fact, in Algorithm 2 (respectively in Algorithm 3) we use the fact that P can be cut into non-trivial simplices by considering the convex hull of the set of vertices minus one. This is only true because P is assumed to be simplicial (since our toric Fano variety is $\mathbb{Q}$-factorial) and has Picard rank two: ignoring one vertex gives a non-trivial polytope in $\mathbb{R}^D$ with $D+1$ vertices, so it is a simplex. Consider now a $\mathbb{Q}$-factorial Picard rank three variety. This corresponds to a simplicial polytope P with $D+3$ vertices in $\mathbb{R}^D$. We can imagine ignoring one of the vertices, obtaining a new polytope Q with $D+2$ vertices in $\mathbb{R}^D$. Then, we could imagine traingulating it as in the Picard rank two case, by ignoring one vertex at a time and taking the convex hull. However, to do so we would need Q to be simplicial, but there is no reason why that would be the case given only that P is simplicial. For example consider the product $\mathbb{P}^1 \times \mathbb{P}^1 \times \mathbb{P}^1$, a three-dimensional toric Fano variety of Picard rank three, as in Example 6.1.8.

Example 6.1.8. Consider $\mathbb{P}^1 \times \mathbb{P}^1 \times \mathbb{P}^1$, which can be given as the following weight matrix

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} .$$

We can associate to it the polytope $P \subset \mathbb{R}^3$ in Figure 6.1 with vertices

$$e_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} , \quad e_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} , \quad e_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} ,$$
$$e_4 = \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix} , \quad e_5 = \begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix} , \quad e_6 = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} .$$

Note that even if P is simplicial, removing any of the vertices e_i and considering the convex hull of the remaining vertices is a pyramid with a square base, which is not simplicial. Therefore, removing a subset of the vertices in turn (like we could do in the case of Proposition 6.1.3) does not always yield a triangulation into simplices. For example, removing $\{e_1, e_2\}$ gives a degenerate simplex.

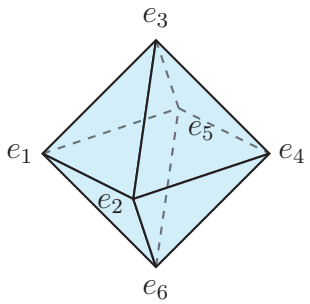


Figure 6.1: A polytope associated to $\mathbb{P}^1 \times \mathbb{P}^1 \times \mathbb{P}^1$.

Therefore, products of weighted projective spaces are problematic, but their terminality check is trivial (since the product is terminal if each weighted projective space is terminal – so we fall back on Proposition 1.1.3). Unfortunately, these are not the only bad examples, meaning that this methodology for checking terminal singularities does not generalise in a straightforward way, but it will require an extra check that the triangulation of the polytope obtained by removing certain vertices is given by non-degenerate simplices.

6.2 From period sequence to weight matrix

In this section, we describe ongoing work in designing an algorithm to determine whether a sequence can arise as the period sequence of a $\mathbb{Q}$-factorial toric Fano variety, returning its weight matrix in case it can. This is meaningful in the context of the mirror symmetry for Fano varieties picture described in Section 1.5. In fact, the aim there is to classify Fano varieties by looking at their mirror partner, Laurent polynomials. Recall, that a Fano variety and a Laurent polynomial are said to be mirror partners if there is an equality between regularized quantum period of the variety and classical period of the polynomial. Therefore,

if we are given a Laurent polynomial and its classical period, we could ask: does it correspond to a toric Fano variety? And if yes, which one? We will not answer this question completely in this section, but we take a first step by looking at the weighted projective spaces and a special case for Picard rank two toric Fano varieties. Note that the desired output of the algorithm is not only a positive or negative answer, but it should produce possible weight matrices that describe completely the geometry of the plausible toric Fano variety that corresponds to the given sequence.

The inputs will be a truncated sequence $(p_i)_{i=0}^n$ (for some $n > 0$) and an integer D, which correspond to the dimension. Note that in Chapters 3 and 4, we have showed that there is a strong relationship between the asymptotic behaviour of the regularized period and the dimension of the corresponding variety. Therefore, it is a reasonable assumption to expect that we could include the expected dimension of the variety as part of the inputs, since it can be obtained from the truncated sequence, if it corresponds to the period sequence of a toric Fano variety.

We implement such an algorithm for weighted projective spaces and in a special case of the Picard rank two case in SageMath [The23], which can be found at [Ven24b].

6.2.1 Weighted projective spaces

Recall from (1.5.1) that the coefficients of the regularized quantum period for the weighted projective space $\mathbb{P}(a_0, \ldots, a_D)$ are

$$c_d = \begin{cases} \frac{d!}{\prod_{i=0}^D (ka_i)!} & \text{if } a = kd \text{ for some } k \in \mathbb{N} \\ 0 & \text{otherwise} \end{cases}$$

where $a = \sum_{i=0}^D a_i$. Therefore, if we are given a truncated sequence $p = (p_i)_{i=0}^n$, it can arise as the period sequence of a D-dimensional weighted projective space only if $p_i = c_i$ for all $i = 0, \ldots, n$, for some weights $(a_0, \ldots, a_D)$. This is the case only if

$$\{i \mid p_i \neq 0\} \subset \{ka \mid k \in \mathbb{N}\}$$

for $a = \min\{i \mid p_i \neq 0 \text{ and } i \neq 0\}$. If this is satisfied, then the period sequence can correspond to a weighted projective space and a is the possible sum of its weights.

To obtain the possible sets of weights, first consider p_a. The aim is to find $(a_0, \ldots, a_D)$ such that

$$\prod_{i=0}^D a_i! = \frac{a!}{p_a} \in \mathbb{Z}\,. \tag{6.2.1}$$

Finding some weights that satisfy the above is equivalent to factorising $a!/p_a$ into products of factorials. We outline the steps of a recursive algorithm to factorise an integer into a product of factorials (which are neither 0 nor 1) in Algorithm 5. If this returns any possible sets of weights, then we can check if any of them can give rise to a weighted projective space of dimension D: they must have $D + 1$ factor or less. If they have less than $D + 1$ factors we can pad the list with 1's, since $1! = 1$, so it does not change the equality (6.2.1).

Lastly, we need to check whether the remaining sets of weights have the correct sum, which must be $a = \min\{i \mid p_i \neq 0 \text{ and } i \neq 0\}$ as specified above. If any possible set of weights survive, we compute their period sequence $(c_i)_{i=0}^n$ and compare it to $(p_i)_{i=0}^n$, and return only those for which they coincide. These steps are outlined in Algorithm 4 and made explicit in the following example.

Example 6.2.1. Consider the sequence

$$p = (1, 0, 0, 6, 0, 0, 90, 0, 0, 1680) .$$

We have seen in Example 1.5.5 that these are the initial coefficients of the regularized quantum period of $\mathbb{P}^2$. Assuming we did not know that, let us find the weights of a weighted projective space directly whose period sequence $(c_i)_{i=0}^9$ coincides with $(p_i)_{i=0}^9$. Firstly, note that

$$\{i \in \mathbb{N} \mid p_i \neq 0\} \subset \{3k \mid k \in \mathbb{N}\} .$$

Therefore, if this sequence arises as the period sequence of a weighted projective space, we must have $a = 3$. So if $p_3 = c_3$ for some $(a_0, \ldots, a_D)$ we must have

$$\prod_{i=0}^{D} a_i! = \frac{3!}{p_3} = 1 .$$

Therefore the only possibility is $(a_0, a_1, a_2) = (1, 1, 1)$, since we have the extra condition that $\sum_{i=0}^{D} a_i = a = 3$ and none of the a_i's can be zero. Comparing the coefficients of the regularized quantum period obtained by this choice of weights to $(p)_{i=0}^9$ we conclude that this sequence can as the period sequence of $\mathbb{P}^2$.

6.2.2 Picard rank two

We now have a process to check if a truncated sequence can arise as the period sequence for a D-dimensional weighted projective space. We would like to generalise this to higher Picard rank toric Fano varieties: in this thesis we will consider only the Picard rank two case. Recall that a Picard rank two toric Fano variety can be described by a weight matrix

$$\begin{bmatrix} a_1 & \cdots & a_N \\ b_1 & \cdots & b_N \end{bmatrix} \tag{6.2.2}$$

with $a_i, b_i \in \mathbb{Z}_{\geq 0}$ and $N = D + 2$. Let $a = \sum_{i=1}^N a_i$ and $b = \sum_{i=1}^N b_i$. The coefficients of the regularized period sequence of the corresponding toric Fano variety are given as in (1.5.2),

$$c_d = \sum_{\substack{ak+bl=d \\ (k,l)\in C\cap\mathbb{Z}^2}} \frac{d!}{\prod_{i=1}^N (ka_i + lb_i)!} \tag{6.2.3}$$

where

$$C = \{(k, l) \in \mathbb{R}^2 \mid a_i k + b_i l \geq 0 \text{ for all } i = 1, \ldots, N\} . \tag{6.2.4}$$

Algorithm 4: The algorithm computes the weights of possible weighted projective space of dimension D whose period sequence coincides with the truncated input sequence $(p_i)_{i=0}^n$. If no such weighted projective space exists it returns the empty list.

```
Function IsWPS(p = (p_i)_{i=0}^n: list, D: int): list of vectors is
    a ←min{i | p_i ≠ 0 and i ≠ 0};
    weights ←[ ];
    if {ka | k ∈ ℕ} eq {i | p_i ≠ 0} then
        x ←a!/p_a;
        prods_factorial ←CheckProductFactorials(x) # From Algorithm 5;
        for l in prods_factorial do
            if (length(l)≤ D + 1) and (D + 1−length(l) eq a−sum(l)) then
                while length(l) < D + 1 do
                    Append 1 to l ;
                end
                Append l to weights;
            end
        end
    else
        Return weights;
    end
end
```

Assume again that our algorithm input is a truncated sequence $p = (p_i)_{i=0}^n$ and the expected dimension D. We want the output of our algorithm to be a set of possible weight matrices like (6.2.2). However, recall from Section 5.1 that this weight matrices do not uniquely correspond to toric Fano varieties. Therefore, we instead aim to construct an output of the form

$$\begin{bmatrix} a_1 & a_2 & \cdots & a_N \\ 0 & b_2 & \cdots & b_N \end{bmatrix} \tag{6.2.5}$$

with $a \leq b$, $a_N < b_N$, $a_i, b_i \in \mathbb{Z}_{\geq 0}$, and the columns ordered cyclically. Each group orbit in the space of weight matrices has at most two representatives of this form. Recall that this is the standard form that we have introduced in Section 5.1, see (5.1.2). Therefore, the output of the algorithm will be a set of possible weight matrices in standard form, whose truncated period sequence coincides with the input sequence.

Depending on the shape of the standard form, the problem can be easier or harder to tackle. Therefore, we implement a solution only in a special case of Picard rank two varieties, namely when $a_N = 0$ in (6.2.5). In fact, in this case the cone in (6.2.4) is always the positive quadrant, i.e.

$$C = \mathbb{R}^2_{\geq 0} \, ,$$

independently of the other entries the weight matrix. On the other hand, if $a_N \neq 0$ then

$$C = \text{Cone}\left(\left[\begin{smallmatrix} b_N \\ -a_N \end{smallmatrix} \right], \left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix} \right] \right) \, .$$

Algorithm 5: The recursive algorithm returns all the ways a positive integer n can be written as a product of factorials (none of which are 0 or 1).

```
Function CheckProductFactorials(n: int): list of vectors is
    result ←( );
    if n = 1 then
        Return [( )];
    else
        P ←max{p | p is prime and p divides n};
        Q ←min{p | p is prime and p > P};
        for i = 0 ... Q − 1 do
            N ←P + i;
            n ←n/N! ;
            if n ∈ ℕ then
                L ←CheckProductFactorials(n);
                for y in L do
                    Append N to y;
                end
                Append L to result;
            else
                Pass;
            end
        end
    end
    Return result;
end
```

Meaning that the domain for the sum in (6.2.3) depends on the entries of the possible weight matrix. We discuss this condition at the end of the section.

A special case: $a_N = 0$

Given an integer D (the expected dimension) and a truncated sequence $p = (p_i)_{i=0}^n$, we want to check whether it can give rise to the period sequence of a toric Fano variety of Picard rank two, given by a weight matrix as in (6.2.2) with $a_N = 0$. We structure the algorithm in the following way.

1. We compute some possible $a = \sum_{i=1}^N a_i$ and $b = \sum_{i=1}^N b_i$ by looking $\{i \mid p_i \neq 0\}$.
2. By looking at p_a and p_b we obtain a finite list of candidates of for the a_i's and b_i's.
3. By looking at p_{a+b} we understand how to match the possible a_i's and b_i's into a weight matrix.
4. If we have found some possible matching, we compare the resulting truncated period sequences against $(p)_{i=0}^n$ to see which weight matrix is a good candidate.

Let us first informally walk through these steps in an example.

Example 6.2.2. Assume we are given the truncated sequence

$$\begin{aligned} p = (&1, 0, 0, 0, 0, \\ &60, 0, 0, 0, 0, \\ &18900, 0, 0, 0, 0, \\ &46246200, 0, 0, 0, 0, \\ &199910610900, 0, 0, 0, 0, 559045745818560) \, . \end{aligned}$$

We want to check whether it can arise as the period sequence of a toric Fano variety of dimension four and Picard rank two (with the extra assumption that $a_6 = 0$ when the corresponding weight matrix is in standard form). Explicitly, we want to find some $a_i, b_i \in \mathbb{Z}_{\geq 0}$ such that

$$\begin{bmatrix} a_1 & a_2 & a_3 & a_4 & a_5 & 0 \\ 0 & b_2 & b_3 & b_4 & b_5 & b_6 \end{bmatrix}$$

is a weight matrix whose truncated period sequence $(p_i)_{i=0}^{20}$ coincides with $(p)_{i=0}^{20}$. By looking at the pattern of zeros in $(p_i)_{i=0}^{20}$ we note that

$$\{i \mid p_i \neq 0\} \subset \{5k \mid k \in \mathbb{N}\} \, .$$

We note from (6.2.3) that for a period sequence $(c_i)_i$ arising from a toric Fano variety of Picard rank two

$$\{i \mid c_i \neq 0\} = \{ak + bl \mid (k, l) \in \mathbb{Z}^2 \cap C\}$$

where C is as in (6.2.4). Since we are assuming $a_6 = 0$, then $C = \mathbb{R}^2_{\geq 0}$. Therefore,

$$\{i \mid c_i \neq 0\} = \{ak + bl \mid (k, l) \in \mathbb{Z}^2_{\geq 0}\} \, .$$

Therefore, if $(p_i)_{i=0}^{20}$ arises as the period sequence of a toric Fano variety of Picard rank two such that $C = \mathbb{R}^2_{\geq 0}$ then we must have $a = 5$ and $5 \mid b$ (without loss of generalities we have chosen $a \leq b$). Since all non-zero coefficients of $(p_i)_{i=0}^{20}$ have indices divisible by 5 we are unable to identify b precisely: we just know that it is divisible by 5. Hence, we need to try different values of b until we find some a_i's and b_i's that match the original series or we run out of terms of the truncated input sequence.

First, assume $b = 5$. Therefore, let $X = \prod_{i=1}^{6} a_i!$ and $Y = \prod_{i=1}^{6} b_i!$, so that

$$p_5 = 60 = \frac{5!}{X} + \frac{5!}{Y} = \frac{120(X + Y)}{XY}$$

i.e. $XY = 2(X + Y)$. Moreover, we know that both X and Y are products of factorials and they must be less than $5! = 120$, since $\sum_{i=1}^{6} a_i = \sum_{i=1}^{6} b_i = 5$. Therefore, by looking at all possible positive integers that are products of factorials and less than $5!$ we obtain that the only solution is $X = Y = 4 = 2!2!$. This tells us that both the a_i's and the b_i's are equal (as unordered lists) to

$$(2, 2, 1, 0, 0, 0)$$

where we have padded with 1 and 0's to sets of weights of the correct length that sum to 5.

Is this the correct solution? We verify this by looking at the next non-zero term of the input

sequence, p_{10}. Let $Z = \prod_{i=1}^{6}(a_i + b_i)!$, then

$$c_{10} = 18900 = \frac{10!}{\prod_{i=1}^{6}(2a_i)!} + \frac{10!}{\prod_{i=1}^{6}(a_i + b_i)!} + \frac{10!}{\prod_{i=1}^{6}(2b_i)!} = 2\frac{10!}{1152} + \frac{10!}{Z} .$$

Hence, there are two possibilities to write Z as a product of factorials

$$Z = 288 = (3!)^2(2!)^3 = 4!3!2! .$$

For each of these products of factorials we check whether we can match the a_i's and the b_i's such that the sums $a_i + b_i$'s match the single factors. This is not possible for $(3!)^2(2!)^3$, but it is for $4!3!2!$ and the proposed weight matrix is

$$\begin{bmatrix} 1 & 2 & 2 & 0 & 0 & 0 \\ 0 & 1 & 2 & 2 & 0 & 0 \end{bmatrix} .$$

However, we cannot have zero columns, so this is not an admissible solution. Therefore, $b \neq 5$.

Next, let us $b = 10$. So, the coefficient p_5 tells us information only about the a_i's. Explicitly,

$$\prod_{i=1}^{6} a_i! = \frac{5!}{60} = 2 .$$

So by factoring 2 as a product of factorials, and appropriately padding, we obtain that the a_i's are, as an unordered list,

$$[2, 1, 1, 1, 0, 0] .$$

Looking at p_{10} gives us information about the b_i's,

$$p_{10} = 18900 = \frac{10!}{\prod_{i=1}^{6}(2a_i)!} + \frac{10!}{\prod_{i=1}^{6} b_i!} = \frac{10!}{192} + \frac{10!}{\prod_{i=1}^{6} b_i!} = 18900 + \frac{10!}{\prod_{i=1}^{6} b_i!} > 18900$$

which is a contradiction. Hence, $b \neq 10$.

Let us check $b = 15$. The possibilities for the a_i's stay the same, since assuming $b = 15$ does not change the calculation for p_5. Also, p_{10} is not influenced by the b_i's now. Therefore, we just check that our proposed a_i's do not lead to a contradiction, and they do not:

$$18900 = \frac{10!}{\prod_{i=1}^{6}(2a_i)!} = \frac{10!}{4!2!2!2!} = 18900 .$$

Consider p_{15} and let $Z = \prod_{i=1}^{6} b_i!$, then

$$46246200 = \frac{15!}{\prod_{i=1}^{6}(3a_i)!} + \frac{15!}{Z}$$

so

$$Z = 34560 = 6!3!(2!)^3 = 5!(3!)^2(2!)^2 = 6!4!2! = 5!4!3!2! .$$

Excluding the last possibility (since it sums to more than 15), then the b_i's can have the

following values (after padding appropriately)

$$(6,3,2,2,2,0)\,,$$
$$(6,4,2,1,1,1)\,,$$
$$(5,4,3,2,1,0)\,.$$

Looking at p_{20} we can see how to match the a_i's and the b_i's. Let $Z = \prod_{i=1}^{6}(a_i+b_i)!$, then

$$p_{20} = 199910610900 = \frac{20!}{\prod_{i=1}^{6}(4a_i)!} + \frac{20!}{Z}$$

then

$$\begin{aligned} Z = 12441600 &= (5!)^2(3!)^3 2!2! = (5!)^2 4!(3!)^2 = (6!)^2 3!(2!)^2 \\ &= (6!)^2 4! = 6!5!(3!)^2(2!)^2 = 6!5!4!3!\,. \end{aligned}$$

The first product of factorials is not possible because the variety is six-dimensional, while all others are possible. We identify how to match the a_i's and b_i's to give rise to an appropriate weight matrix in Table 6.1.

$\prod_{i=1}^{6}(a_i+b_i)!$	Possible weight matrices		
$6!6!3!(2!)^2$	$\begin{bmatrix}1&1&2&2&0&0\\2&2&4&3&6&1\end{bmatrix}$		
$6!5!4!3!$	$\begin{bmatrix}1&1&2&1&0&0\\0&2&3&5&1&4\end{bmatrix}$	$\begin{bmatrix}1&2&1&1&0&0\\0&2&4&5&1&3\end{bmatrix}$	
	$\begin{bmatrix}1&1&2&1&0&0\\0&2&4&3&1&5\end{bmatrix}$	$\begin{bmatrix}1&1&2&1&0&0\\0&1&4&5&2&3\end{bmatrix}$	

Table 6.1: Possible weight matrices corresponding to the different decompositions of 12441600 as a product of factorials.

We immediately note that the first possibility is not in standard form, so we exclude it. To understand which of the other possibilities is the correct one we compute the period sequence of each one and see which one matches the given input. The only survivor is

$$\begin{bmatrix}1&1&2&1&0&0\\0&2&4&3&1&5\end{bmatrix}\,. \tag{6.2.6}$$

We note that b cannot be greater than 15 since, in that case p_{15} would only depend on the a_i's, but this leads to a contradiction

$$p_{15} = 46246200 \neq 8408400 = \frac{15!}{6!3!3!3!}\,.$$

Therefore, we can conclude that (6.2.6) is the only weight matrix of the desired form that gives us the input sequence.

We outline the precise computational routines below, referencing pseudocode, and include a SageMath implementation in [Ven24b].

- Algorithm 6 is the wrapper function. It computes a and b by looking at $\{i \mid p_i \neq 0\}$. It distinguishes two cases: $\{i \mid p_i \neq 0\} \subset \{mk+nl \mid k,l \in \mathbb{Z}_{\geq 0}\}$ (for some $m \neq n \in \mathbb{N}$) and $\{i \mid p_i \neq 0\} \subset \{mk \mid k \in \mathbb{Z}_{\geq 0}\}$ (for some $m \in \mathbb{N}$). In the first case, since we have that $C = \mathbb{R}^2_{\geq 0}$ by assumption, we can conclude that $\{a,b\} = \{m,n\}$ (without loss of generalities we always choose $a \leq b$). In the second case, we let $a = m$, but we cannot determine the value of b precisely: we can only say that m divides b. In this case, the wrapper function loops through different multiples of m for the value of b until it finds a solution or it runs out of terms in the truncated sequence output. Once it has a value of a and b the wrapper function calls the other routines to compute the possible sets of weight matrices whose period sequence coincides with the input sequence and returns them (only if they are in standard form with $a_N = 0$).
- Algorithm 7 is the heart of the computation. It assumes we have fixed a and b and loops through different terms of the sequence to compute possible entries of the weight matrix a_i's and b_i's. It uses p_a and p_b to compute possible entries for each row in the weight matrix, by calling Algorithm 8. Once it has candidates for the entries of each row of the matrix, it attempts to match them into a matrix using Algorithm 9, by looking at the term p_{a+b}. Once it has a set of possible weight matrices it checks their corresponding period sequences against the input sequence, and returns only those weight matrices for which the input sequence can give rise to its period sequence.
- Algorithm 8 finds for a given input n all possible sets of integers of a specified length and sum whose product of factorials is the input. This is achieved by factorising the input integer into non-trivial products of factorials (using Algorithm 5) and appropriately padding them with 1's and 0's (if possible) to make them of the correct length and sum. Note that unlike the weighted projective space case we are allowed to have zero values.
- Algorithm 9 takes three lists l, m, n as input, and recursively computes all permutations π and σ such that
$$l_i + m_{\pi(i)} = n_{\sigma(i)} \, .$$
The algorithm is inductive on the length of the list, therefore it lifts permutations from S_k to S_{k+1} using Algorithm 10.

Outlook: towards the general case

The above section deals exclusively with the case of a weight matrix in standard form with $a_N = 0$. In this section, we discuss why removing this assumption complicates the approach.

The general case for a Picard rank two variety would consider whether a sequence of number can originate as the period sequence of a toric Fano variety with weight matrix

$$\begin{bmatrix} a_1 & a_2 & \cdots & a_N \\ 0 & b_2 & \cdots & b_N \end{bmatrix} \tag{6.2.7}$$

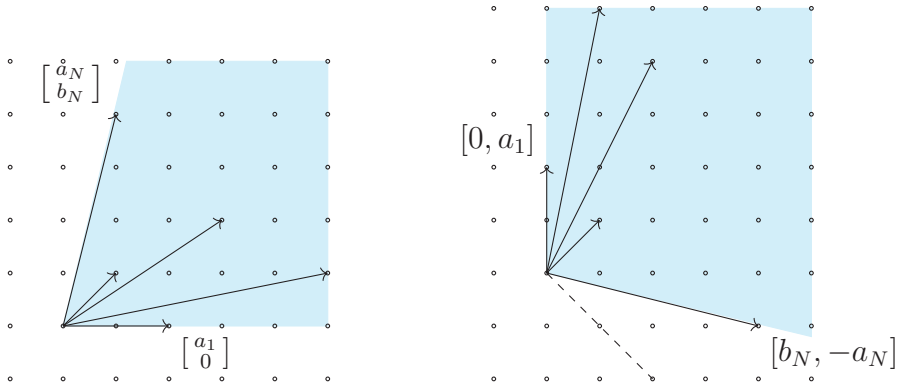
with $a \leq b$, $a_N < b_N$, $a_i, b_i \in \mathbb{Z}_{\geq 0}$, and the columns ordered cyclically. Recall, that the period sequence is defined as

$$c_d = \sum_{\substack{ak+bl=d \\ (k,l)\in C}} \frac{d!}{\prod_{i=1}^{N}(ka_i + lb_i)!}$$

where

$$C = \{(k,l) \in \mathbb{R}^2 \mid a_i k + b_i l \geq 0 \text{ for all } i = 1, \ldots, N\} = \text{Cone}\left(\left[\begin{smallmatrix} b_N \\ -a_N \end{smallmatrix}\right], \left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]\right) ,$$

as we have previously discussed.



(a) Secondary fan. (b) Dual cone

Figure 6.2: Secondary fan (i.e. the columns of the weight matrix) and dual cone for a general weight matrix in standard form corresponding to a Picard rank two variety. Note that since $a_N < b_N$ then C is contained in the convex cone Cone$\left(\left[\begin{smallmatrix} 1 \\ -1 \end{smallmatrix}\right], \left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]\right)$, hence we include the dashed line $y = -x$.

When $a_N = 0$ we have the nice feature that $C = \mathbb{R}^2_{\geq 0}$ independently of the entries of the weight matrix, but when $a_N \neq 0$ then $\mathbb{R}^2_{\geq 0} \subsetneq C$; see Figure 6.2. Note that assumption that the weight matrix is in standard form, so that $a_N < b_N$, implies that $C \subsetneq$ Cone$\left(\left[\begin{smallmatrix} 1 \\ -1 \end{smallmatrix}\right], \left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]\right)$, so at least we know it belongs to a bounded domain.

Not having $C = \mathbb{R}^2_{\geq 0}$ means that we are only able to extract $g = \gcd(a, b)$ from the pattern of zeros of $(p_i)_{i=0}^n$, instead of the values of a and b, like we did in the first step of Algorithm 6. Therefore, a procedure to compute whether a truncated sequence $(p_i)_{i=0}^n$ can arise as the period sequence of a general weight matrix in standard form for a Picard rank two toric Fano variety will have to loop over various possible values of a and b knowing their greatest common factor (just as we did in the previous section when $a \mid b$).

Then, we could imagine running through similar steps to the previous section, but not knowing what C is makes the procedure much more computationally expensive. We could imagine considering p_d where $d = \min\{i > 0 \mid p_i \neq 0\}$. If we assume $p_d = c_d$ for the period sequence $(c_d)_d$ of some weight matrix, then we need to find a_i's and b_i's such that

$$p_d = \sum_{\substack{ak+bl=d \\ (k,l)\in C\cap\mathbb{Z}^2}} \frac{d!}{\prod_{i=1}^{N}(ka_i + lb_i)!} .$$

However, without knowing the cone C, we cannot be sure of what indices this sum is over. We know that $\mathbb{R}^2_{\geq 0} \subset C$, so one could imagine trying to find some candidates by considering $\mathbb{R}^2_{\geq 0} = C$. If we find some solutions, we move to the next non-zero term of the truncated sequence, but if we do not find any solution we consider $C = \text{Cone}\left(\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right], \left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]\right)$ where $\left[\begin{smallmatrix} i \\ j \end{smallmatrix}\right]$ is the next lattice point anticlockwise on the line $ai + bj = d$ outside the positive quadrant. If we still do not find any points, we keep adding anticlockwise lattice points until we hit the line defined by $\left[\begin{smallmatrix} b \\ -a \end{smallmatrix}\right]$ or $\left[\begin{smallmatrix} 1 \\ -1 \end{smallmatrix}\right]$. This procedure would go through all possibilities, but it is much more computationally expensive than the case when $a_N = 0$. We leave as future work implementing this case and studying whether an exhaustive search of the possibilities can be avoided.

Algorithm 6: Wrapper function to compute the weight matrix as in (6.2.2) of possible Picard rank two toric Fano variety of dimension D whose period sequence coincides with the truncated input sequence $(p_i)_{i=0}^n$. If no such weight matrix exists it returns the empty list.

Function `WeightsCalculator`$(p = (p_i)_{i=0}^n$*: list,* D*: int*$)$*: list of matrices* **is**
  `non-zero` $\leftarrow \{i \in \mathbb{N} \mid p_i \neq 0\}$;
  $a \leftarrow \min\{i \in$ `non-zero` $\mid i \neq 0\}$;
  try:
    $b \leftarrow \min\{i \in$ `non-zero` $\mid a \nmid i\}$;
    `weights` $\leftarrow$`ComputeWeights`(p, D, a, b) # From Algorithm 7;
  catch $\{i \in$ `non-zero` $\mid a \nmid i\} = \varnothing$**:**
    $b \leftarrow a$;
    `weights`$\leftarrow$`ComputeWeights`(p, D, a, b) # From Algorithm 7;
    while `weights` $= [(\,), (\,)]$ **do**
      `weights` $\leftarrow$`ComputeWeights`(p, D, a, b) # From Algorithm 7;
      $b \leftarrow b + a$;
    end
  end
  Return $\{w \in$ `weights` $\mid w$ is in standard form with $a_N = 0\}$;
end

Algorithm 7: The algorithm computes the possible weights knowing the period sequence, the dimension, and (a, b) – the sum of the weight matrix columns.

Function `ComputeWeights`($p = (p_i)_{i=0}^n$*: list, D: int, a: int, b: int*)*: list of matrices* **is**
  `w_unordered` ←[ ];
  `non-zero` ←$\{i \in \mathbb{N} \mid p_i \neq 0\}$;
  if b *not eq* a **then**
    `a_poss` ←`PossibleProducts`$(a, D+2, a!/p_a)$ # From Algorithm 8;
    if $a \nmid b$ **then**
      `b_poss` ←`PossibleProducts`$(b, D+2, b!/p_b)$ # From Algorithm 8;
      `w_unordered` ←`w_unordered` + `zip`(`a_poss`, `b_poss`);
    else
      for x *in* `a_poss` **do**
        M ←$b!/(p_b - b!/\prod_i (ba_i/a)!)$;
        `b_poss` ←`PossibleProducts`$(b, D+2, M)$ # From Algorithm 8;
        `w_unordered` ←`w_unordered` + [(x, y) for y in `b_poss`] ;
      end
    end
  else
    `prod_factorials` ←$\{n \in \mathbb{N} \mid n < a!$ and n is a product of factorials$\}$;
    for x *and* y *in* `prod_factorials` **do**
      if $a!(1/x + 1/y)$ *eq* p_a **then**
        `a_poss` ←`PossibleProducts`$(a, D+2, x)$ # From Algorithm 8;
        `b_poss` ←`PossibleProducts`$(a, D+2, y)$ # From Algorithm 8;
        `w_unordered` ←`w_unordered` + `zip`(`a_poss`, `b_poss`);
      end
    end
  end
  for w *in* `w_unordered` **do**
    `a_list`, `b_list` ←first row of w, second row of w ;
    if $a \mid b$ **then**
      k ←b/a+1 ;
      M ←$(a+b)!/(p_{a+b} - (a+b)!/\prod_i (ka_i)!)$;
    else
      M ←$(a+b)!/p_{a+b}$;
    end
    `ab_list` ←`PossibleProducts`$(a+b, D+2, M)$ # From Algorithm 8;
    if `ab_list` *is not empty* **then**
      for (π, σ) *in* `ComputePerms`(`a_list`, `ab_list`, `ab_list`) **do**
        `b_list_perm` ←π(`b_list`)
      end
      Append [`a_list`, `b_list_perm`] to `weights`;
    end
  end
  Return $\{w \in$ `weights` $\mid$ the truncated period sequence of w equals $(p_i)_{i=0}^n\}$;
end

Algorithm 8: The algorithm returns all sets of integers of length L and sum s whose product of factorials is n.

```
Function PossibleProducts (s: int, L: int, n: int): list of vectors is
    weights ←CheckProductFactorials(n) # From Algorithm 5;
    filtered_weights ←[ ];
    for l in weights do
        if sum(l)≤ a then
            for i = 1,...,sum(l)−a do
                Append 1 to l;
            end
            if length(l)≤ L then
                for i = 1,...,length(l)−L do
                    Append 0 to l;
                end
                Append l to filtered_weights;
            end
        end
    end
    Return filtered_weights;
end
```

Algorithm 9: The algorithm returns all possible permutations π and σ such that $l_i + m_{\pi(i)} = n_{\sigma(i)}$. Note that n_{-1} stands for the last element in the list n.

```
Function ComputePerms (l: list, m: list, n: list): list of permutations is
    if length(n) eq 1 then
        Return [(0), (0)]
    else
        L ←{(i, j) | l_i + m_j = n_{-1}};
        for (i, j) in L do
            l^i←(l with the ith element removed);
            m^j←(m with the jth element removed);
            n^{-1}←(n with the last element removed);
            for (π, σ) in ComputePerms(l^i,m^j,n^{-1}) do
                Append (Lift(π, i, j), Lift(σ, i, −1)) to result # From
                  Algorithm 10;
            end
        end
        Return result;
    end
end
```

Algorithm 10: The algorithm lifts a permutation from S_k to S_{k+1} so that i is mapped to j.

```
1  Function Lift (perm: list, i: int, j: int): list is
2      l ←length(perm);
3      new_perm ←a list of zeros of length l + 1;
4      for k = 0 . . . l − 1 do
5          if k = i then
6              new_perm[k] ←j ;
7          else if perm[k] < j and k < i then
8              new_perm[k] ←perm[k];
9          else if perm[k] ≥ j and k < i then
10             new_perm[k] ←perm[k]+1;
11         else if perm[k − 1] < j and k > i then
12             new_perm[k] ←perm[k − 1];
13         else if perm[k − 1] ≥ j and k > i then
14             new_perm[k] ←perm[k − 1]+1;
15     end
16     Return new_perm;
17 end
```

7 Conclusion and Outlook

We conclude this thesis with some reflections on integration of machine learning with pure mathematics. We discuss some positive and negative aspects of our approach and sketch some natural next steps.

Beyond Fano Varieties

This work is well-positioned in the context of experimental and computational mathematics. The use of *large-scale computations* in pure mathematics is a relatively recent area of research that is becoming more popular with the improvement of computational efficiency. The machine learning methodology we have illustrated in this thesis complements these recent efforts in experimental mathematics by providing a range of additional *model-blind* tools.

Therefore, the AI-enhanced experimental workflow proposed in this thesis for algebraic geometry is already applicable in those settings that employ large computations and experiments. For example, number theory has a long history of experimental mathematics and computations (recall the computations that first led to the formulation of the Birch and Swinnerton-Dyer Conjecture [BSD63]), and there are already some examples of applications of AI in this area [HLO22, HLOP22]. We also recall from the Introduction and Section 2.5 some examples of research areas that benefit from the availability of datasets mathematical objects, e.g. [BKnt, APC+16, KS00, CCN+85, Cre16]. Data generation and data availability is the biggest constraint in applying machine learning methods to pure mathematics, therefore we expect areas that already have large datasets available to be the most amenable to this methodology.

Nevertheless, in our work we have not used a pre-existing dataset, but instead we have generated our own data, using computer algebra software (namely Magma [BCP97] and SageMath [The23]). Similar specialised software would be used by researchers generating their own datasets. Most machine learning libraries (PyTorch [PGM+19], TensorFlow [AAB+15], JAX [BFH+18]) are integrated in Python. However, with the exception of SageMath which is written in python and OSCAR [OSC24] which is written in Julia, most other computer algebra systems are based on their own language (e.g. Magma [BCP97], Mathematica [Inc23] and Macaulay2 [GS]). A next step to make this workflow accessible to more pure mathematics researchers would be integrating these systems with existing machine learning libraries more efficiently.

Sampling

As discussed above (and a common aspect of this work with [DVB+21]), the datasets used in this thesis were generated with the specific problems we wanted to investigate in mind. There were not pre-existing and complete datasets of mathematical objects. Dealing with this type of data generation opens up the problem of random sampling from an unknown distribution in the context of mathematical data. It raises questions like '*what is a random algebraic variety*?', '*what is a random knot diagram?*', '*is randomly generated data the correct framework to work with?*'. While these issues might encourage us to concentrate on complete datasets rather than random samples, we remark that working on the latter has some benefits. In fact, carrying out machine learning experiments on datasets that contain complete classifications of mathematical objects might highlight novel relationships. However, these might not be of great use, since they have been found by working on a dataset where all the objects and properties have already been computed. Working with a random sample from an unknown classification could lead to finding new relationships that might help understand the unknown classification of objects better leading to new mathematics. However, randomly sampling in this context might lead to bias and formulating incorrect conjectures. In this section, we discuss the dangers of sampling bias when it comes to generating mathematical datasets for machine learning.

In the first part of [DVB+21], the authors carry out some experiments to predict relationships between knot invariants using machine learning. They work on a database made up of three different datasets: all knots up to 16 crossings (from the Regina census [Bur20]); random knots diagrams of 80 crossings; knots obtained as closures of certain braids. The latter dataset of knots was included to disprove the first conjecture which was suggested by the machine learning analysis, which turned out to be incorrect. The authors remarked that the original training data did not include these 'non-generic' examples of knots, which are actually counterexamples to the conjecture. Therefore, this shows how randomness or genericity might not be the correct property to require when constructing training data. Usual applications of machine learning in scientific domains call for algorithms to be *robust to outliers*, i.e. they must be able to ignore noisy data samples. However, unless some randomness is included in the data generation steps, there are no outliers in data coming from mathematical constructions: every sample is correct and interesting. This leads to two issues. Firstly, machine learning algorithms might ignore interesting examples in mathematical data, because they do not *fit* with the generic objects: however, these might be counterexamples to false conjectures, and therefore they should definitely not be ignored. Secondly, while generating random mathematical objects can produce samples which are generic, they might not be actually *representative*, since they might not include less common – but still interesting – examples.

In this thesis, we also had to make some decision regarding sampling criteria, but we encountered issues with a slightly different flavour. When looking at properties of weight matrices in Chapter 5, we generated random weight matrices in a specific *standard form* (a choice of group orbit representative), by uniformly sampling each entry of the matrix given some fixed bounds. This is an easy construction, however it does not lead to a true random sample of weight matrices (and therefore of algebraic varieties). This is made clear in Figure 5.3b and Figure 5.7. These pictures highlight how the dataset contains a cluster of varieties with Fano index around 35. Therefore, our sampling criterion cannot

be truly random, since there is no reason to have more varieties cluster around that Fano index value. As we make clear in Section 5.3.2, this is a result of the bound on the entries of the weight matrices that we have chosen in order to generate each weight matrix *randomly* (by choosing each entry uniformly at random within some specified bounds).

Our issues with sampling are somewhat different from those from [DVB+21]. We are not excluding a family of important samples because they are less common, but we are imposing a sampling criterion which is clearly not uniformly random. However, we remark that while our criterion is not truly random (and therefore it comes with certain downsides), it is easily formulated and implemented. When generating data we must pay attention to the trade-off between generating generic data and computable data. There might be contexts where we are not able to formulate a computable criterion to generate truly random samples. In these cases, choosing a non-representative sampling criterion which is systematically computable is not necessarily the worst choice. In fact, conducting machine learning experiments on biased data can still highlight a criterion that can be proven rigorously in general or for a larger class of objects. A small example of this is the experiments in Chapter 3, where we also dealt with weight matrices as a representation of toric Fano varieties. When generating the data we impose that each weight matrix must contain an identity block (see Remark 3.1.1 for details). This restriction was done for computational reasons, and does not produce a representative data sample (since not all weight matrices can be assumed to have an identity block). However, the machine learning highlighted a behaviour that was then rigorously proven for much more general objects in Chapter 4. As long as we are aware of what kind of bias they are introducing when generating data, being unable to generate a truly random sample of mathematical objects should not be an obstacle from using a machine learning approach. As we have seen, this type of studies can still highlight non-trivial relationships that are then proven rigorously.

Data Representation and Group Actions

As highlighted in Chapter 5, our representation of toric Fano varieties via *weight matrices* comes equipped with two group actions: an action of the permutation group S_N acting by permutation of the columns, and an action of $\mathrm{GL}_2(\mathbb{Z})$, that acts on the left by matrix multiplication. In the context of mathematical data, objects often come with presentations that benefit (or suffer) exactly from this: they come equipped with a group action that changes the representation but leaves the underlying mathematical object the same. Therefore, we need a machine learning framework that takes such group actions into account in some way.

The study of equivariant and invariant machine learning brings approaches to deal with exactly this problem. For example, one can build a model taking the symmetry into account when designing its architecture [HGLBR18, ZKR+17]. On the other hand, we could add synthetic examples obtained by choosing a lot of different representatives of the same symmetry class [CDL20, KSH12]. In our context in Chapter 5, we addressed this problem using *fundamental domain projection*, meaning that we choose a representative from each group orbit which has a specific form. This is a cheap preprocessing step that helped us address the action of the group $\mathrm{GL}_2(\mathbb{Z})$: this is an infinite discrete group. There is no architecture that has been designed to be invariant with respect to this group action, and it

is unclear how to meaningfully augment the data, since the group is infinite. However, fundamental domain projection comes with its drawbacks. In fact, for more complicated groups it might be impossible to find a suitable representative of each group orbit. Moreover, the choice of a certain orbit representative complicates the sampling discussion from the previous section: is sampling uniformly from the space of representatives equal to sampling uniformly from the underlying distribution? Future research will concentrate on benchmarking different equivariant and invariant machine learning approaches against each other, in order to understand which one is better suited to which group action and task.

AI for Maths and LLMs

Given the context of this thesis, it is important to mention the recent work in using Large Language Models (LLMs) to produce correct mathematical proofs of statements. Recent results in this direction have been produced using LLMs in conjunction with interactive theorem provers, like LEAN [YSG+24] and Isabelle [JLHW21]. This type of paradigm is very different to the workflow we have covered in this thesis. The idea is to train an LLM on a library of formal proofs in order to generate new proofs in that formal language, given a formalised statement as a prompt. Since proof assistants benefit from a *logic checker*, which can check whether the produced proof is mathematically correct. We can imagine using the feedback from the logic checker in some reinforcement learning loop, which improves the output of the LLM and eventually produces a correct proof.

Ideas in this fashion are very promising and there have already been exciting and unexpected results in this direction; see [AAt24, TL24, PHZ+22, XRS+24]. Most of these examples focused on dealing with problems from the International Mathematical Olympiad (IMO), and are not yet addressing research-level mathematics. Part of the reason is the lack of a large corpus of formalised proofs for these models to learn from. Such a corpus is expensive to obtain (not only time-consuming, but it requires a lot of mathematical expertise to produce), and we can expect that this will be a considerable obstacle for bridging the gap from IMO problems to open questions found in the mathematics literature.

The use of more standard machine learning tools (as we have done in this work) has a lower entry level (both in terms of machine learning expertise and computational requirements) than LLMs and can be already integrated into a research-level workflow. At the same time, these are not competing research directions, since we have been focusing more on *discovering* new mathematics with the aid of machine learning, rather than *proving* it. In the future, we can imagine using machine learning on large datasets of mathematical objects to suggest conjectures, which can then be proven rigorously and automatically using an LLM trained on formal proofs.

Mathematics as Testing Ground

Finally, we conclude by describing a promising flavour of future research, which we have not addressed in this thesis. The study of mathematical data using a machine learning

approach is not only beneficial to address problems in mathematics, but can also drive machine learning research.

As we have pointed out above, data originating from mathematical constructions has no noise nor outliers. It is *perfect data*. Using machine learning tools on mathematical data can aid our understanding of three fundamental problems in the application of artificial intelligence: sampling, representation, and explainability. Note how these topics mirror the themes we have recalled in the previous sections. This is a statement to the close interactions between pure mathematics and machine learning. Let us sketch what these problems are and how mathematics is a good setting to address them.

Sampling bias occurs when the training data of a model is not uniform: some members of a population might have a higher or lower chance to be represented in the data. The model might erroneously attribute this bias in the sampling to a novel phenomenon rather than noisy data generation. When applying machine learning to mathematical data there is the unique advantage of having complete control over the data generation and sampling. In fact, data collection in other subjects might be unclear and not well-documented for the machine learning practitioner, while also often being expensive and time-consuming. On the other hand, when building mathematical datasets the sampling criterion is described precisely and the data generation is algorithmic. Therefore, mathematical datasets present the unique opportunity to study how sampling bias affects the performance of machine learning models and how to recognise it in a highly controlled environment.

Similarly, mathematical datasets provide a unique testing ground when it comes to studying how machine learning algorithms deal with symmetries in the data representations (symmetries are often described by group actions). Mathematical objects are rich with complicated symmetries and, unlike real-world data (where a '9' can either be a nine or a rotated six), are completely regular with respect to them. Therefore, mathematical data is useful to benchmark current approaches to symmetries in the constructions of AI systems. Moreover, it gives us the opportunity to develop new methods, since the mathematical objects might come with group actions which are not found commonly in real-world data.

Lastly, one of the main open questions in both AI applications to mathematics and pure AI research is explainability. AI systems are often built on *black box models* (such as neural networks). Explainability – being able to fully understand the reasons for an AI prediction – is often out of reach, but it is a fundamental feature required in many AI applications, especially in the context of AI safety. Understanding the inner workings of AI systems is vital to determining in which context they can be safely deployed. This is, of course, also desirable in the context of AI applications to mathematics, where the ultimate aim is to understand what the statement the model is suggesting in order to prove it. Again, mathematical data is the perfect setting to probe questions related to explainability for AI, since – by its very nature – precise rules must underpin any signal uncovered by machine learning. Therefore, mathematical datasets are the ideal testing ground to evaluate and develop explainable models.

Abbreviations and Acronyms

Abbreviation	Definition
(A)NN	(Artificial) Neural Network
CPU	Central Processing Unit
`DSdim_wps`	Dataset of $150\,000$ weighted projective spaces
`DSdim_rk2`	Dataset of $200\,000$ Picard rank two toric Fano varieties
`DSdim_rk2_cut`	`DSdim_rk2` filtered to include only varieties with dimension higher than six
`DSterm_10M`	Balanced dataset of 10 million $\mathbb{Q}$-factorial terminal and non-terminal toric Fano varieties with weights bounded by 7
`DSterm_20M`	Balanced dataset of 20 million $\mathbb{Q}$-factorial terminal and non-terminal toric Fano varieties with weights bounded by 10
`DSterm_100M`	Dataset of 100 million probable $\mathbb{Q}$-Fano toric varieties of Picard rank two and dimension eight
HPC	High Performance Computing
GIT	Geometric Invariant Theory
LeakyReLU	Leaky Rectified Linear Unit
LOWLESS	Locally Weighted Scatterplot Smoothing
ML	Machine Learning
MLP	Multi-Layer Perceptron
MMP	Minimal Model Program
MSE	Mean Squared Error
PCA	Principal Component Analysis
$\mathbb{Q}$-Fano	$\mathbb{Q}$-factorial Fano variety with at worst terminal singularities
ReLU	Rectified Linear Unit
SHAP	SHapley Additive exPlanations
SVM	Support Vector Machine

List of Figures

List of Tables

List of Algorithms

Bibliography

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[AAt24] AlphaProof and AlphaGeometry teams. AI achieves silver-medal standard solving International Mathematical Olympiad problems. Online, 2024. `https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/`.

[Aga18] Abien F. Agarap. Deep learning using rectified linear units (ReLU). arXiv:1803.08375 [cs.NE], 2018.

[AGW18] Philipp Arras, Antonella Grassi, and Timo Weigand. Terminal singularities, Milnor numbers, and matter in F-theory. *Journal of Geometry and Physics*, 123:71–97, 2018.

[AHDM78] Michael F. Atiyah, Nigel J. Hitchin, Vladimir G. Drinfeld, and Yuri I. Manin. Construction of instantons. *Phys. Lett. A*, 65(3):185–187, 1978.

[Ahm17] Hamid Ahmadinezhad. On pliability of del Pezzo fibrations and Cox rings. *J. Reine Angew. Math.*, 723:101–125, 2017.

[APC+16] Jeffrey Adams, Annegret Paul, Ran Cui, Susana Salamanca-Riba, Peter Trapa, Marc van Leeuwen, and David Vogan. Atlas of Lie groups and representations. Online, 2016. `http://www.liegroups.org`.

[APS23] Benjamin Aslan, Daniel Platt, and David Sheard. Group invariant machine learning by fundamental domain projections. In *NeurIPS Workshop on Symmetry and Geometry in Neural Representations*, pages 181–218. PMLR, 2023.

[Bat81] Victor V. Batyrev. Toric Fano threefolds. *Izv. Akad. Nauk SSSR Ser. Mat.*, 45(4):704–717, 927, 1981.

[Bat91] Victor V. Batyrev. On the classification of smooth projective toric varieties. *Tohoku Math. J.* (2), 43(4):569–585, 1991.

[Bat99] Victor V. Batyrev. On the classification of toric Fano 4-folds. volume 94, pages 1021–1050. 1999. Algebraic geometry, 9.

[BBD+22] Charles Blundell, Lars Buesing, Alex Davies, Petar Veličković, and Geordie Williamson. Towards combinatorial invariance for Kazhdan–Lusztig polynomials. *Representation Theory of the American Mathematical Society*, 26(37):1145–1191, 2022.

[BCP97] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).

[BCZ04] Gavin Brown, Alessio Corti, and Francesco Zucconi. Birational geometry of 3-fold Mori fibre spaces. In *The Fano Conference*, pages 235–275. Univ. Torino, Turin, 2004.

[Ber16] Robert J. Berman. K-polystability of $\mathbb{Q}$-Fano varieties admitting Kähler–Einstein metrics. *Invent. Math.*, 203(3):973–1025, 2016. doi:10.1007/s00222-015-0607-7.

[BFH+18] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[BHH+22] Jiakang Bao, Yang-Hui He, Edward Hirst, Johannes Hofscheier, Alexander M. Kasprzyk, and Suvajit Majumder. Hilbert series, machine learning, and applications to physics. *Physics Letters B*, 827:136966, 2022.

[BHH+23] Jiakang Bao, Yang-Hui He, Edward Hirst, Johannes Hofscheier, Alexander M. Kasprzyk, and Suvajit Majumder. Polytopes and machine learning. *International Journal of Data Science in the Mathematical Sciences*, 2(1):181–211, 2023.

[BHH+24] Per Berglund, Yang-Hui He, Elli Heyes, Edward Hirst, Vishnu Jejjala, and Andre Lukas. New Calabi–Yau manifolds from genetic algorithms. *Physics Letters B*, page 138504, 2024.

[BHJM18] Kieran Bull, Yang-Hui He, Vishnu Jejjala, and Challenger Mishra. Machine learning CICY threefolds. *Physics Letters B*, 785:65–72, 2018.

[Bir21] Caucher Birkar. Singularities of linear systems and boundedness of Fano varieties. *Ann. of Math.* (2), 193(2):347–405, 2021.

[BKnt] Gavin Brown and Alexander M. Kasprzyk. The graded ring database. Online, 2007–present. `http://www.grdb.co.uk`.

[BSD63] Bryan J. Birch and H. Peter F. Swinnerton-Dyer. Notes on elliptic curves. I. *J. Reine Angew. Math.*, 212:7–25, 1963.

[Bur20] Benjamin A. Burton. The next 350 million knots. In *36th International Symposium on Computational Geometry*, volume 164 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 25, 17. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2020.

[CA03] Kai Lai Chung and Farid AitSahlia. *Elementary probability theory*. Undergraduate Texts in Mathematics. Springer-Verlag, New York, fourth edition, 2003.

[Cas21] Paolo Cascini. New directions in the minimal model program. *Boll. Unione Mat. Ital.*, 14(1):179–190, 2021.

[CCG+13] Tom Coates, Alessio Corti, Sergey Galkin, Vasily Golyshev, and Alexander M. Kasprzyk. Mirror symmetry and Fano manifolds. In *European Congress of Mathematics*, pages 285–300. Eur. Math. Soc., Zürich, 2013.

[CCGK16] Tom Coates, Alessio Corti, Sergey Galkin, and Alexander M. Kasprzyk. Quantum periods for 3-dimensional Fano manifolds. *Geom. Topol.*, 20(1):103–256, 2016.

[CCIT15] Tom Coates, Alessio Corti, Hiroshi Iritani, and Hsian-Hua Tseng. A mirror theorem for toric stacks. *Compos. Math.*, 151(10):1878–1912, 2015.

[CCN+85] John H. Conway, Robert T. Curtis, Simon P. Norton, Richard A. Parker, and Robert A. Wilson. $\mathbb{ATLAS}$ *of finite groups*. Oxford University Press, Eynsham, 1985. Maximal subgroups and ordinary characters for simple groups, With computational assistance from J. G. Thackray.

[CDL20] Shuxiao Chen, Edgar Dobriban, and Jane H. Lee. A group-theoretic framework for data augmentation. *The Journal of Machine Learning Research*, 21(1):9885–9955, 2020.

[CHK22] Tom Coates, Liana Heuberger, and Alexander M. Kasprzyk. Mirror symmetry, Laurent inversion and the classification of $\mathbb{Q}$-Fano threefolds. arXiv:2210.07328 [math.AG], 2022.

[CHK23] Tom Coates, Johannes Hofscheier, and Alexander M. Kasprzyk. Machine learning the dimension of a polytope. In *Machine learning in pure mathematics and theoretical physics*, pages 85–104. World Scientific, 2023.

[CHSW85] Philip Candelas, Gary T. Horowitz, Andrew Strominger, and Edward Witten. Vacuum configurations for superstrings. *Nuclear Phys. B*, 258(1):46–74, 1985. doi:10.1016/0550-3213(85)90602-9.

[CK22] Tom Coates and Alexander M. Kasprzyk. Databases of quantum periods for Fano manifolds. *Scientific Data*, 9(1):163, 2022.

[CKP19] Tom Coates, Alexander M. Kasprzyk, and Thomas Prince. Laurent inversion. *Pure Appl. Math. Q.*, 15(4):1135–1179, 2019.

[CKPT21] Tom Coates, Alexander M. Kasprzyk, Giuseppe Pitton, and Ketil Tveiten. Maximally mutable Laurent polynomials. *Proc. A.*, 477(2254):Paper No. 20210584, 21, 2021.

[CKV22a] Tom Coates, Alexander M. Kasprzyk, and Sara Veneziale. A dataset of 150000 terminal weighted projective spaces, 2022. doi:10.5281/zenodo.5790079.

[CKV22b] Tom Coates, Alexander M. Kasprzyk, and Sara Veneziale. A dataset of 200000 terminal toric varieties of Picard rank 2, 2022. doi:10.5281/zenodo.5790096.

[CKV22c] Tom Coates, Alexander M. Kasprzyk, and Sara Veneziale. Supporting code to *Machine learning the dimension of a Fano variety*. `https://bitbucket.org/fanosearch/mldim`, 2022.

[CKV23a] Tom Coates, Alexander M. Kasprzyk, and Sara Veneziale. A dataset of 8-dimensional $\mathbb{Q}$-factorial Fano toric varieties of Picard rank 2, 2023. doi:10.5281/zenodo.10046893.

[CKV23b] Tom Coates, Alexander M. Kasprzyk, and Sara Veneziale. Machine learning the dimension of a Fano variety. *Nature Communications*, 14(5526), 2023.

[CKV23c] Tom Coates, Alexander M. Kasprzyk, and Sara Veneziale. Supporting code to *Machine learning detects terminal singularities*. `https://bitbucket.org/fanosearch/ml_terminality`, 2023.

[CKV24] Tom Coates, Alexander M. Kasprzyk, and Sara Veneziale. Machine learning detects terminal singularities. *Advances in Neural Information Processing Systems*, 36, 2024.

[CLS11] David A. Cox, John B. Little, and Henry K. Schenck. *Toric varieties*, volume 124 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 2011.

[Cre16] John Cremona. The L-functions and modular forms database project. *Found. Comput. Math.*, 16(6):1541–1553, 2016.

[DH98] Igor V. Dolgachev and Yi Hu. Variation of geometric invariant theory quotients. *Inst. Hautes Études Sci. Publ. Math.*, (87):5–56, 1998. With an appendix by Nicolas Ressayre.

[DJLT21] Alex Davies, András Juhász, Marc Lackenby, and Nenad Tomasev. The signature and cusp geometry of hyperbolic knots. arXiv:2111.15323 [math.GT], 2021.

[DLQ22] Michael Douglas, Subramanian Lakshminarasimhan, and Yidi Qi. Numerical Calabi–Yau metrics from holomorphic networks. In *Mathematical and Scientific Machine Learning*, pages 223–252. PMLR, 2022.

[DP87] Pasquale Del Pezzo. Sulle superficie dell'n^{mo} ordine immerse nello spazio ad n dimensioni. *Rend. del Circolo Mat. di Palermo*, 1:241–255, 1887.

[DVB+21] Alex Davies, Petar Veličković, Lars Buesing, Sam Blackwell, Daniel Zheng, Nenad Tomašev, Richard Tanburn, Peter Battaglia, Charles Blundell, András Juhász, Marc Lackenby, Geordie Williamson, Demis Hassabis, and Pushmeet Kohli. Advancing mathematics by guiding human intuition with AI. *Nature*, 600:70–74, 2021.

[EF21] Harold Erbin and Riccardo Finotello. Machine learning for complete intersection Calabi–Yau manifolds: a methodological study. *Phys. Rev. D*, 103(12):Paper No. 126014, 40, 2021.

[EO13] European Organization For Nuclear Research and OpenAIRE. Zenodo, 2013.

[ERSS05] Nicholas Eriksson, Kristian Ranestad, Bernd Sturmfels, and Seth Sullivant. Phylogenetic algebraic geometry. In *Projective varieties with unexpected properties*, pages 237–255. Walter de Gruyter, Berlin, 2005.

[Fan47] Gino Fano. Nuove ricerche sulle varietà algebriche a tre dimensioni a curvesezioni canoniche. *Pont. Acad. Sci. Comment.*, 11:635–720, 1947.

[FRPM06] Ioannis A. Fotiou, Philipp Rostalski, Pablo A. Parrilo, and Manfred Morari. Parametric optimization and optimal control using algebraic geometry methods. *Internat. J. Control*, 79(11):1340–1358, 2006.

[Ful93] William Fulton. *Introduction to toric varieties*, volume 131 of *Annals of Mathematics Studies*. Princeton University Press, Princeton, NJ, 1993. The William H. Roever Lectures in Geometry.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[Gér22] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. " O'Reilly Media, Inc.", 2022.

[GHMR23] Sergei Gukov, James Halverson, Ciprian Manolescu, and Fabian Ruehle. Searching for ribbons with machine learning. arXiv:2304.09304 [math.GT], 2023.

[Giv96] Alexander B. Givental. Equivariant Gromov–Witten invariants. *Internat. Math. Res. Notices*, (13):613–663, 1996.

[Giv98] Alexander B. Givental. A mirror theorem for toric complete intersections. In *Topological field theory, primitive forms and related topics (Kyoto, 1996)*, volume 160 of *Progr. Math.*, pages 141–175. Birkhäuser Boston, Boston, MA, 1998.

[Giv01] Alexander B. Givental. Semisimple Frobenius structures at higher genus. *Internat. Math. Res. Notices*, (23):1265–1286, 2001.

[GK13] Roland Grinis and Alexander M. Kasprzyk. Normal forms of convex lattice polytopes. arXiv:1301.6641 [math.CO], 2013.

[Gne18] Boris V. Gnedenko. *Theory of probability*. Routledge, 2018.

[Gre97] Brian R. Greene. String theory on Calabi–Yau manifolds. In *Fields, strings and duality (Boulder, CO, 1996)*, pages 543–726. World Sci. Publ., River Edge, NJ, 1997.

[GS] Daniel R. Grayson and Michael E. Stillman. Macaulay2, a software system for research in algebraic geometry. Available at `http://www2.macaulay2.com`.

[GS19] Mark Gross and Bernd Siebert. Intrinsic Mirror Symmetry. arXiv:1909.07649 [math.AG], 2019.

[He23] Yang-Hui He. Machine-learning mathematical structures. *International Journal of Data Science in the Mathematical Sciences*, 1(1):23–47, 2023.

[Heu22] Liana Heuberger. $\mathbb{Q}$-fano threefolds and Laurent inversion. arXiv:2202.04184 [math.AG], 2022.

[HEW+14] Zongyan Huang, Matthew England, David Wilson, James H. Davenport, Lawrence C. Paulson, and James Bridge. Applying machine learning to the problem of choosing a heuristic to select the variable ordering for cylindrical algebraic decomposition. In *Intelligent Computer Mathematics: International Conference, CICM 2014, Coimbra, Portugal, July 7-11, 2014. Proceedings*, pages 92–107. Springer, 2014.

[HGLBR18] Jason Hartford, Devon Graham, Kevin Leyton-Brown, and Siamak Ravanbakhsh. Deep models of interactions across sets. In *International Conference on Machine Learning*, pages 1909–1918. PMLR, 2018.

[HKK+03] Kentaro Hori, Sheldon Katz, Albrecht Klemm, Rahul Pandharipande, Richard Thomas, Cumrun Vafa, Ravi Vakil, and Eric Zaslow. *Mirror symmetry*, volume 1 of *Clay Mathematics Monographs*. American Mathematical Society, Providence, RI; Clay Mathematics Institute, Cambridge, MA, 2003. With a preface by Vafa.

[HKS22] Kathryn Heal, Avinash Kulkarni, and Emre Can Sertöz. Deep learning Gauss–Manin connections. *Advances in Applied Clifford Algebras*, 32(2):24, 2022.

[HLO22] Yang-Hui He, Kyu-Hwan Lee, and Thomas Oliver. Machine-learning the Sato–Tate conjecture. *Journal of Symbolic Computation*, 111:61–72, 2022.

[HLOP22] Yang-Hui He, Kyu-Hwan Lee, Thomas Oliver, and Alexey Pozdnyakov. Murmurations of elliptic curves. arXiv:2204.10140 [math.NT], 2022.

[Hug20] Mark C. Hughes. A neural network approach to predicting and computing knot invariants. *Journal of Knot Theory and Its Ramifications*, 29(03):2050005, 2020.

[IDS23] Shima Imani, Liang Du, and Harsh Shrivastava. MathPrompter: Mathematical reasoning using large language models. In *2023 Meeting of the Association for Computational Linguistics*, 2023.

[Inc23] Wolfram Research, Inc. *Mathematica, Version 14.1*, 2023. `https://www.wolfram.com/mathematica`.

[Isk77] V. A. Iskovskih. Fano threefolds. I. *Izv. Akad. Nauk SSSR Ser. Mat.*, 41(3):516–562, 717, 1977.

[Isk78] V. A. Iskovskih. Fano threefolds. II. *Izv. Akad. Nauk SSSR Ser. Mat.*, 42(3):506–549, 1978.

[Isk79] V. A. Iskovskih. Anticanonical models of three-dimensional algebraic varieties. In *Current problems in mathematics, Vol. 12* (*Russian*), pages 59–157, 239 (loose errata). VINITI, Moscow, 1979.

[JCB+23] Helen Jenne, Herman Chau, Davis Brown, Jackson Warley, Timothy Doster, and Henry Kvinge. Can we count on deep learning: Exploring and characterizing combinatorial structures using machine learning. In *The 3rd Workshop on Mathematical Reasoning and AI at NeurIPS'23*, 2023.

[JLHW21] Albert Q. Jiang, Wenda Li, Jesse M. Han, and Yuhuai Wu. Lisa: Language models of Isabelle proofs. In *6th Conference on Artificial Intelligence and Theorem Proving*, pages 378–392, 2021.

[Kal19] Elana Kalashnikov. Four-dimensional Fano quiver flag zero loci. *Proc. Royal Society A.*, 475(2225):20180791, 23, 2019.

[Kas06] Alexander M. Kasprzyk. Toric Fano three-folds with terminal singularities. *Tohoku Math. J.* (2), 58(1):101–121, 2006.

[Kas09] Alexander M. Kasprzyk. Bounds on fake weighted projective space. *Kodai Math. J.*, 32(2):197–208, 2009.

[Kas13] Alexander M. Kasprzyk. Classifying terminal weighted projective space. arXiv:1304.3029 [math.AG], 2013.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv:1412.6980 [cs.LG], 2014.

[KIK+23] Hiroshi Kera, Yuki Ishihara, Yuta Kambe, Tristan Vaccon, and Kazuhiro Yokoyama. Learning to compute Gröbner bases. arXiv:2311.12904 [math.AC], 2023.

[KM98] János Kollár and Shigefumi Mori. *Birational geometry of algebraic varieties*, volume 134 of *Cambridge Tracts in Mathematics*. Cambridge University Press, Cambridge, 1998.

[KMM92] János Kollár, Yoichi Miyaoka, and Shigefumi Mori. Rational connectedness and boundedness of Fano manifolds. *J. Differential Geom.*, 36(3):765–779, 1992.

[KN09] Maximilian Kreuzer and Benjamin Nill. Classification of toric Fano 5-folds. *Adv. Geom.*, 9(1):85–97, 2009.

[Kol87] János Kollár. The structure of algebraic threefolds: an introduction to Mori's program. *Bull. Amer. Math. Soc.* (*N.S.*), 17(2):211–273, 1987.

[KS00] Maximilian Kreuzer and Harald Skarke. Complete classification of reflexive polyhedra in four dimensions. *Adv. Theor. Math. Phys.*, 4(6):1209–1230, 2000.

[KS04] Maximilian Kreuzer and Harald Skarke. PALP: a package for analysing lattice polytopes with applications to toric geometry. *Comput. Phys. Comm.*, 157(1):87–106, 2004.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25, 2012.

[LC20] Guillaume Lample and François Charton. Deep learning for symbolic mathematics. In *International Conference on Learning Representations*, 2020.

[LCB10] Yann LeCun, Corinna Cortes, and Christopher J. C. Burges. MNIST handwritten digit database. Online, 2010. ATT Labs, `http://yann.lecun.com/exdb/mnist`.

[Lee23] Kyu-Hwan Lee. Data–scientific study of Kronecker coefficients. arXiv:2310.17906 [math.RT], 2023.

[LJR+20] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-Tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. *Proceedings of Machine Learning and Systems*, 2:230–246, 2020.

[LL17] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, 30, 2017.

[LLN+18] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E. Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. arXiv:1807.05118 [cs.LG], 2018.

[LNA17] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A Python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017.

[MHN13] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, page 3. Atlanta, GA, 2013.

[MM03] Shigefumi Mori and Shigeru Mukai. Erratum: "Classification of Fano 3-folds with $B_2 \geq 2$". *Manuscripta Math.*, 110(3):407, 2003.

[MM82] Shigefumi Mori and Shigeru Mukai. Classification of Fano 3-folds with $B_2 \geq 2$. *Manuscripta Math.*, 36(2):147–162, 1981/82.

[Mor85] Shigefumi Mori. On 3-dimensional terminal singularities. *Nagoya Math. J.*, 98:43–66, 1985.

[MPP23] Jelena Mojsilović, Dylan Peifer, and Sonja Petrović. Learning a performance metric of Buchberger's algorithm. *Involve, a Journal of Mathematics*, 16(2):227–248, 2023.

[NX09] Harald Niederreiter and Chaoping Xing. *Algebraic geometry in coding theory and cryptography*. Princeton University Press, Princeton, NJ, 2009.

[Øbr07] Mikkel Øbro. An algorithm for the classification of smooth Fano polytopes. arXiv:0704.0049 [math.CO], 2007.

[OSC24] OSCAR – Open Source Computer Algebra Research system, 2024.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Bai Junjie, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[PHZ+22] Stanislas Polu, Jesse M. Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya Sutskever. Formal mathematics statement curriculum learning. arXiv:2202.01344 [cs.LG], 2022.

[Pol05] Joseph Polchinski. *String theory. Vol. II*. Cambridge Monographs on Mathematical Physics. Cambridge University Press, Cambridge, 2005. Superstring theory and beyond, Reprint of 2003 edition.

[PSHL20] Dylan Peifer, Michael Stillman, and Daniel Halpern-Leistner. Learning selection strategies in Buchberger's algorithm. In *International Conference on Machine Learning*, pages 7575–7585. PMLR, 2020.

[PVG+11] Fabina Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Rei80] Miles Reid. Canonical 3-folds. In *Journées de Géometrie Algébrique d'Angers, Juillet 1979/Algebraic Geometry, Angers, 1979*, pages 273–310. Sijthoff & Noordhoff, Alphen aan den Rijn—Germantown, Md., 1980.

[Rei87] Miles Reid. Young person's guide to canonical singularities. In *Algebraic geometry, Bowdoin, 1985 (Brunswick, Maine, 1985)*, volume 46 of *Proc. Sympos. Pure Math.*, pages 345–414. Amer. Math. Soc., Providence, RI, 1987.

[RPBN+24] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

[RS20] Maithra Raghu and Eric Schmidt. A survey of deep learning for scientific discovery. arXiv:2003.11755 [cs.LG], 2020.

[Sat00] Hiroshi Sato. Toward the classification of higher-dimensional toric Fano varieties. *Tohoku Math. J. (2)*, 52(3):383–413, 2000.

[Sha53] Lloyd S. Shapley. A value for n-person games. In *Contributions to the theory of games, vol. 2*, volume no. 28 of *Ann. of Math. Stud.*, pages 307–317. Princeton Univ. Press, Princeton, NJ, 1953.

[SHM+16] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[SW22] Nawaz Sultani and Rachel Webb. Subtleties of Quantum Lefshetz without Convexity. arXiv:2208.08987 [math.AG], 2022.

[SYZ96] Andrew Strominger, Shing-Tung Yau, and Eric Zaslow. Mirror symmetry is T-duality. *Nuclear Phys. B*, 479(1-2):243–259, 1996.

[Tel12] Constantin Teleman. The structure of 2D semi-simple field theories. *Invent. Math.*, 188(3):525–588, 2012.

[The23] The Sage Developers. *SageMath, the Sage Mathematics Software System* (*Version 9.8*), 2023. `https://www.sagemath.org`.

[TL24] Trieu Trinh and Thang Luong. AlphaGeometry: An Olympiad-level AI system for geometry. Online, 2024. `https://deepmind.google/discover/blog/alphageometry-an-olympiad-level-ai-system-for-geometry/`.

[Ven] Sara Veneziale. Asymptotics of the regularized quantum period. In preparation.

[Ven24a] Sara Veneziale. The ML4Maths Pipeline. *Der Computeralgebra–Rundbrief*, 74, 2024.

[Ven24b] Sara Veneziale. Supporting code to *Fano Varieties and Machine Learning*. `https://bitbucket.org/sveneziale/thesis_code`, 2024.

[vLvdG88] Jacobus H. van Lint and Gerard van der Geer. *Introduction to coding theory and algebraic geometry*, volume 12 of *DMV Seminar*. Birkhäuser Verlag, Basel, 1988.

[Wag21] Adam Zsolt Wagner. Constructions in combinatorics via neural networks. arXiv:2104.14516 [math.CO], 2021.

[Wan19] Jun Wang. A mirror theorem for Gromov–Witten theory without convexity. arXiv:1910.14440 [math.AG], 2019.

[WDL22] Yue Wu and Jesús A. De Loera. Turning mathematics problems into games: Reinforcement learning and Gröbner bases together solve integer feasibility problems. arXiv:2208.12191 [cs.LG], 2022.

[Wil23] Geordie Williamson. Is deep learning a useful tool for the pure mathematician? arXiv:2304.12602 [math.RT], 2023.

[WW82] Keiichi Watanabe and Masayuki Watanabe. The classification of Fano 3-folds with torus embeddings. *Tokyo J. Math.*, 5(1):37–48, 1982.

[XRS+24] Huajian Xin, Z. Z. Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, Wenjun Gao, Qihao Zhu, Dejian Yang, Zhibin Gou, Z. F. Wu, Fuli Luo, and Chong Ruan. DeepSeek-Prover-V1. 5: Harnessing Proof Assistant Feedback for Reinforcement Learning and Monte-Carlo Tree Search. arXiv:2408.08152 [cs.CL], 2024.

[YSG+24] Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J. Prenger, and Animashree Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models. *Advances in Neural Information Processing Systems*, 36, 2024.

[ZKR+17] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R. Salakhutdinov, and Alexander J. Smola. Deep sets. *Advances in Neural Information Processing Systems*, 30, 2017.