Applications of Machine Learning to Fano Polygons.

by Pratyush Joshi,

Submitted to The University of Nottingham

in September 2024

in partial fulfilment of the conditions for the award of the degree of

Master of Science in Data Science

Student ID: 20569233

Email ID: psxpj7@nottingham.ac.uk

Supervised By: Dr. Daniel Cavey

I declare that this dissertation is all my own work, except as indicated in the text

Contents:

Table of Figures:

Table of Tables:

Abstract

This dissertation explores the application of machine learning to the study of Fano lattice polygons, focusing on predicting T-singularities with the help of Erhart numbers—a significant challenge in algebraic geometry. Traditional combinatorial and geometric methods often struggle with the high-dimensional complexity of these problems. By understanding machine learning models such as Logistic Regression, Random Forests, Support Vector Classification (SVC), and Gradient Boosting, this research aims to enhance the prediction accuracy of number of T-singularities within Fano lattice polygons.

The study emphasizes feature engineering and dimensionality reduction techniques, including Polynomial Features and Principal Component Analysis (PCA), to capture complex relationships within the data. It also discusses the importance of model validation through cross-validation techniques to ensure robust and generalizable predictions. The integration of machine learning not only provides new insights into the structure of Fano lattice polygons but also offers a more scalable and flexible approach compared to traditional methods.

Experimental results demonstrate that these machine learning models can significantly improve the prediction of number of T-singularities, with models like Logistic Regression , Random Forests and Gradient Boosting showing particularly strong performance. The findings suggest that the combination of algebraic geometry and machine learning can lead to more accurate and interpretable models, potentially advancing the broader field of algebraic geometry and its applications in areas such as combinatorial design and theoretical physics.

Acknowledgements

With deep appreciation, I would like to thank Dr. Daniel Cavey for all his help and support during my research. His domain knowledge, perceptive criticism, and guidance have been essential in molding my comprehension and have substantially enhanced the accomplishment of my thesis.

I also want to sincerely thank my family for their everlasting encouragement and support. My strength has come from their understanding and support, and I sincerely appreciate everything they have done for me.

Lastly, I would like to express my gratitude to all my professors and fellow students for their help and advice during my MSc in Data Science course. I sincerely appreciate all your assistance, which has been important in helping me meet my academic objectives.

Chapter 1: Introduction

1.1 Motivation

The study of Fano lattice polygons is pivotal in algebraic geometry due to their deep connections with combinatorics, topology, and number theory. These polygons, a special class of lattice polytopes, are significant because of their association with Fano varieties, which are essential in the classification of algebraic varieties within the Minimal Model Program (MMP) (Kawamata, Matsuda, & Matsuki, 1988).

The geometric and combinatorial properties of Fano lattice polygons are vital for both theoretical exploration and practical applications. These polygons are directly linked to the classification of toric Fano varieties, a task that becomes increasingly complex as the dimension of the polytopes increases. Specifically, understanding T-singularities—special types of terminal quotienT-singularities—within these polygons is a significant mathematical challenge. T-singularities are essential for understanding the birational geometry of algebraic varieties and play a vital role in the study of mirror symmetry, a fundamental concept in string theory and theoretical physics (Mori & Reid, 1987).

Traditional methods for analyzing T-singularities in Fano lattice polygons have relied primarily on combinatorial and geometric techniques. The combinatorial complexity of these problems can lead to computational difficulties, limiting the accuracy and efficiency of T-singularity properties (Dais, 2002). Furthermore, the reliance on specific geometric insights can restrict the generalizability of these methods to new or unseen lattice polytopes (Kasprzyk, 2010).

Given these challenges, there is a growing interest in applying machine learning techniques to the study of Fano lattice polygons. Machine learning, known for its ability to model complex relationships and handle high-dimensional data, offers a promising insight to guide traditional methods. By training models on existing data, we believe it is possible to predict the presence of T-singularities in new lattice polytopes, potentially leading to more accurate and generalizable results (Bishop, 2006).

The motivation for this dissertation arises from the desire to work at this intersection of traditional mathematics and modern machine learning approaches. By understanding the strengths of both fields, this research aims to develop novel predictive models that enhance our understanding of T-singularities in Fano lattice polygons. The potential impact of this research extends beyond algebraic geometry, with implications for related fields such as combinatorial design, coding theory, and theoretical physics (Cox, Little, & Schenck, 2011).

1.2 Research Objectives

The primary objective of this dissertation is to explore the relationship between Ehrhart numbers and the number of T-singularities in Fano lattice polygons. This exploration will involve the application of various machine learning techniques, including Polynomial Features, Principal Component Analysis

(PCA), Logistic Regression, Random Forests, Gradient Boosting, and Support Vector Classification (SVC). Each method has been selected for its potential to address specific challenges associated with the prediction of T-singularities.

- **Objective 1:** Collect and Prepare the Dataset: Gather a comprehensive dataset of Fano lattice polygons, ensuring that it includes relevant geometric properties and T-singularity counts. Develop the underlying algorithm necessary to compute the Ehrhart polynomial coefficients and identify T-singularities. This objective is essential as it establishes the foundation for subsequent analyses and model development.
- **Objective 2:** Explore the Relationship Between Ehrhart Numbers and T-Singularity Counts: Analyze the dataset to investigate potential correlations between the coefficients of the Ehrhart polynomial and the presence or number of T-singularities. This exploration will help identify patterns that could serve as predictors of T-singularities, thereby contributing to a deeper understanding of the geometric structure of Fano lattice polygons.
- **Objective 3: Develop and Interpret Feature Engineering Techniques:** Apply feature engineering techniques, such as Polynomial Features and Principal Component Analysis (PCA), to enhance the predictive models' accuracy and interpretability. This objective focuses on creating new features that may reveal hidden patterns in the data, thereby improving the models' ability to predict T-singularities and offering insights into the underlying geometric structures.
- **Objective 4: Evaluate Machine Learning Techniques for Predicting T-Singularities:** Train and evaluate several machine learning models, including Logistic Regression, Random Forests, Gradient Boosting, and SVC, on the dataset. The models will be assessed using metrics such as accuracy, precision, recall, and F1-score. The aim is to identify the most effective and interpretable model for predicting T-singularities, contributing to the practical application of machine learning in this area.

1.3 Scope of the Study

This dissertation focuses on the application of machine learning to predict the number of T-singularities in Fano lattice polygons, using Ehrhart numbers as key features. The scope of the research undertaken is limited to two-dimensional Fano lattice polygons due to their tractability, and the easy combinatorial interpretation of T-singularities in this dimension. The research is also limited to the number of vertices as 3 and 4 for polygon for better understanding. Although the study could be extended to higher dimensions and a greater number of vertices, this research will concentrate on the foundational understanding of the problem in two dimensions, providing a basis for future explorations (Fulton, 1993).

1.4 Structure of the Dissertation

The dissertation is structured as follows:

- **Chapter 2: Literature Review Mathematical Foundation** – This chapter reviews the core mathematical concepts relevant to the study, including Fano lattice polygons, T-singularities, and Ehrhart theory. It explores the geometric properties of Fano polygons and their significance in algebraic geometry, particularly in the context of classification problems.

- **Chapter 3: Literature Review - Machine Learning Applications -** This chapter explores the use of machine learning in algebraic geometry, focusing on models like Logistic Regression, Random Forests, and SVC for predicting T-singularities. It also discusses feature engineering and evaluation metrics, providing a foundation for applying these techniques to the collected mathematical data.
- **Chapter 4: Methodology -** This chapter outlines the research methods, beginning with data collection and preprocessing of Fano lattice polygons. It details the algorithmic approach to computing Ehrhart polynomials and identifying T-singularities, followed by a discussion of the feature engineering process. Finally, the chapter describes the machine learning models used, including the training, evaluation, and comparison of their performance.
- **Chapter 5: Experimental Setup -** This chapter describes the experimental setup, which includes using Jupyter Notebook. It details the use of Python and libraries like Pandas, NumPy, and Scikit-learn for data handling, machine learning, and feature engineering. The setup also involves data preprocessing, model training, and evaluation with cross-validation, and employs Matplotlib and Seaborn for result visualization.
- **Chapter 5: Results and Discussion -** This chapter presents the findings of the study, including the relationships identified between Ehrhart polynomial coefficients and T-singularity counts. It analyzes the performance of the machine learning models, interpreting the significance of key features and discussing their implications for understanding the geometric structures of Fano polygons. The chapter integrates these results to provide insights into the broader context of algebraic geometry.
- **Chapter 6: Conclusion -** The final chapter summarizes the study's findings, highlighting their contributions to algebraic geometry and the prediction of T-singularities. It addresses the study's limitations and suggests directions for future research, particularly the potential for further developing machine learning approaches to explore other complex geometric structures.

Chapter 2: Literature Review - Mathematical Foundation

2.1 Introduction

This literature review explores essential combinatorial and algebraic geometry concepts vital for understanding the basics of Fano lattice polygons, T-singularities and Ehrhart theory. These concepts provide a foundational framework for examining the intricate relationships between these geometric objects.

Fano lattice polygons are central to understanding specific geometric structures in algebraic geometry, particularly in relation to Fano varieties. These polygons, defined by their convex shape and vertices with coprime integer coordinates, offer insights into the classification of algebraic varieties, which is a core challenge in the field. Their connection to Fano varieties, characterized by a positive anticanonical divisor, makes them essential for exploring the underlying geometric properties that define these varieties.

The relationship between Fano polygons and toric varieties is another key area of study. Toric varieties, which are built from lattice polytopes, demonstrate a duality with these polygons that plays a significant role in understanding mirror symmetry. This duality highlights the deep connections between combinatorial structures and geometric features, bridging concepts in algebraic geometry and offering new perspectives on classification problems.

Additionally, the study of T-singularities associated with Fano polygons deepens our comprehension of the geometric and topological characteristics of these objects. T-singularities occur on the edges of Fano polygons and are defined by the lengths and heights of these edges. By examining these singularities, we gain valuable insights into the more intricate features of Fano varieties, which are vital for their classification and analysis.

While traditional approaches have laid the groundwork for understanding these concepts, contemporary methods like Ehrhart theory further illuminate the combinatorial aspects of Fano polygons. By exploring how the number of lattice points changes in scaled versions of these polygons, Ehrhart theory provides a more nuanced understanding of their geometric properties. This combinatorial approach complements the geometric analysis, offering a well-rounded perspective on the study of Fano polygons.

By integrating these diverse areas of research, this review seeks to deepen our understanding of Fano polygons and their associated singularities. This knowledge is essential for both theoretical advancements and practical applications in mathematics, offering new perspectives and tools for future exploration in the dynamic field of algebraic geometry.

2.2 Lattices and Lattice Polygons

2.2.1 Definition and Importance of Lattices

For the purposes of this dissertation, a lattice in R^2 is defined as a discrete subset that is isomorphic to Z^2. Specifically, in the context of this work, we focus on the two-dimensional case where n=2. This implies that the lattice is a copy of Z^2 embedded in $R^{2.}$

2.2.2 Lattice Polygons

A **lattice polygon** is a convex polygon in the plane where all vertices are lattice points—points with integer coordinates.

Basic Properties:

1. **Vertices:** The vertices of a lattice polygon are integer-coordinate points that determine the shape and size of the polygon.
2. **Edges:** The edges are line segments connecting consecutive vertices. They may contain additional lattice points between the endpoints.
3. **Interior Points:** These are lattice points strictly inside the polygon, excluding those on the boundary.
4. **Boundary Points:** These are lattice points located on the edges of the polygon.

Example: Consider the lattice polygon defined by the vertices (0,0), (2,0) and (0,2). This is a lattice polygon because its vertices are all lattice points.

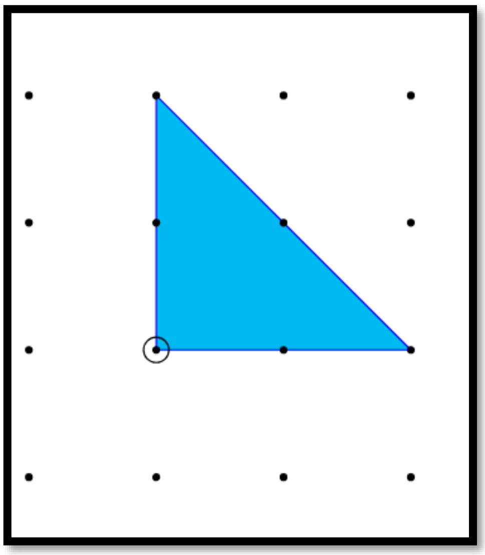


Fig 1: Simple Lattice Polygon

2.3 Fano Lattice Polygons

2.3.1 Definition and Properties

Fano lattice polygons are a specialized class of lattice polygons. The polygon is convex, meaning that any line segment connecting two points within the polygon remains entirely inside it. These polygons are particularly important in algebraic geometry because they correspond to **Fano varieties**, which are special types of geometric shapes that have a rich and well-behaved structure. Fano lattice polygons help in understanding and classifying these geometric shapes, offering insights into their properties [Fulton, 1993].

Let P be a convex lattice polygon in R^2. The polygon P has the origin as one of its interior points, and its vertices are located at the smallest possible lattice points on their respective rays from the origin.

Then P is a Fano lattice polygon if it satisfies the following conditions:

1.Origin as an Interior Point: (0,0) ∈Int(P):

Here, Int(P) denotes the set of all interior lattice points of the polygon P. This condition ensures that the origin lies strictly inside the polygon and not on its boundary. Importantly, this also implies that P must be a two-dimensional polygon; otherwise, the origin could only be a boundary point or lie outside the polygon entirely.

2.Vertices are Primitive Lattice Points:

Each vertex v∈V(P) of the polygon P is a primitive lattice point, meaning that the greatest common divisor (GCD) of its coordinates is 1. This condition ensures that the vertices do not have any common factor greater than 1, thus maintaining the simplicity of the lattice structure.

2.3.2 Examples of Fano Lattice Polygons

1. Non-Fano Polygon Example: Triangle with Vertices at (2,0), (0,2), (-2,0)

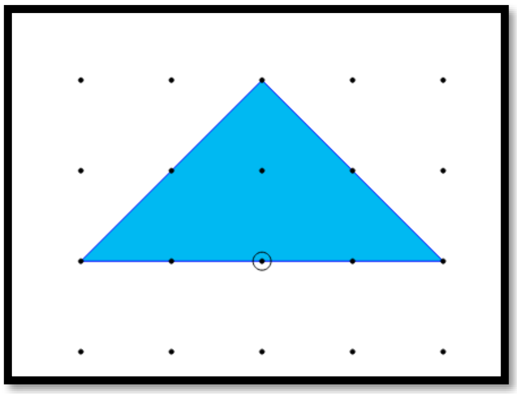


Fig 2: Graphical Representation of Non- Fano Lattice Polygon

Step 1: Identify the Vertices

The vertices of the triangle are (2,0), (0,2), and (-2,0).

Step 2: Compute the GCD for Each Vertex

- For the vertex (2,0): Compute gcd(2,0). The result is 2.
- For the vertex (0,2): Compute gcd(0,2). The result is 2.
- For the vertex (-2,0): Compute gcd(-2,0). The result is 2.

Step 3: Determine Fano Polygon Status

- Since the gcd of the coordinates for vertices (2,0), (0,2), and (-2,0) is greater than 1, the polygon does not satisfy the condition that all vertex coordinates must have a gcd of 1. Additionally, the origin is not inside the triangle, this makes it as a non-Fano polygon due to the non-primitive vertices and origin not inside condition.

Step 4: Conclusion

- **This triangle is not a Fano polygon** because the vertices are not primitive lattice points (their gcd is not 1). The origins lie on the line segment, the non-primitive nature of the vertices and origin condition disqualifies it from being a Fano polygon.

2. Non-Fano Polygon Example: Triangle with Vertices at (1,0), (0,1), and (2, -1)

Step 1: Identify the Vertices

The vertices of the triangle are (1,0), (0,1), and (2,-1).

Step 2: Compute the GCD for Each Vertex

- For the vertex (1,0): Compute gcd(1,0). The result is 1.
- For the vertex (0,1): Compute gcd(0,1). The result is 1.
- For the vertex (2, -1):Compute gcd(2,-1). The result is 1.

Step 3: Determine Fano Polygon Status

- Since the gcd of the coordinates for all vertices is 1, this polygon meets the condition that all vertex coordinates are primitive lattice points. However, the origin **(0,0)** is not strictly in the interior of this triangle, as it lies on the line segment joining two vertices.

Step 4: Conclusion

- **This triangle is not a Fano polygon** because, although the vertices are primitive (gcd = 1), the origin is not in the strict interior of the polygon. The origin lies on the boundary of the polygon, which violates the first condition for Fano polygons.

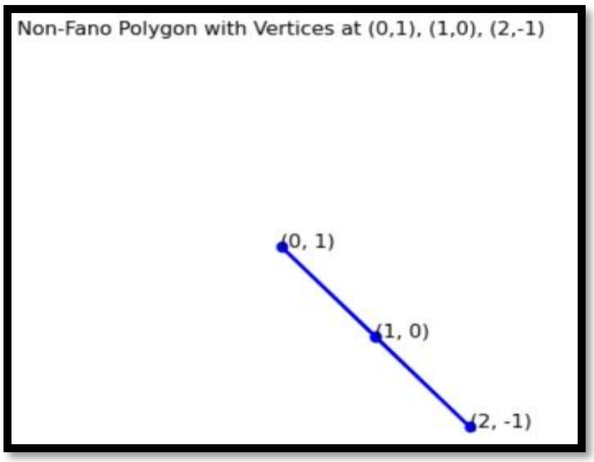


Fig 3: Graphical Representation of Non- Fano Lattice Polygon

3.Fano Polygon Example: Triangle with Vertices at (1,0), (0,1) and (−1, −1).

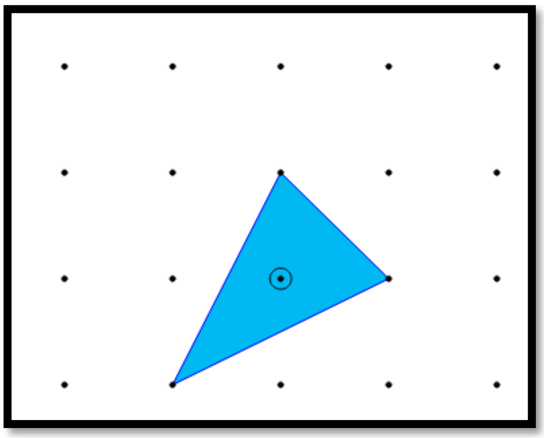


Fig 4: Graphical Representation of Fano Lattice Polygon

Step 1: Identify the Vertices

- The vertices of the triangle are (1,0), (0,1), and (-1, -1).

Step 2: Compute the GCD for Each Vertex

- For the vertex (1,0):
  - Compute gcd(1,0). The result is 1.
- For the vertex (0,1):
  - Compute gcd(0,1). The result is 1.
- For the vertex (-1, -1):
  - Compute gcd(−1,−1). The result is 1.

Step 3: Determine Fano Polygon Status

· Criterion **1: Origin as an Interior Point**

- The first criterion for a polygon to be considered a Fano polygon is that the origin (0,0) must lie strictly within the interior of the polygon.
- In this example: If we draw the triangle with vertices at (1,0), (0,1), and (-1,-1), the origin (0,0) is indeed located inside the triangle. This satisfies the first criterion, as the origin does not lie on any edge or vertex of the triangle but is strictly within its interior.

· Criterion **2: Vertices are Primitive Lattice Points**

- The second criterion is that all vertices of the polygon must be primitive lattice points, meaning the greatest common divisor (GCD) of the coordinates of each vertex is 1.
- In this example: As computed in Step 2, the GCD for each vertex's coordinates is 1, confirming that all vertices are primitive lattice points.

Step 4: Conclusion

This triangle is a Fano polygon because it satisfies both key criteria:

- The origin (0,0) is strictly inside the polygon, satisfying the first criterion.
- All vertices (1,0), (0,1), and (-1, -1) are primitive lattice points with a GCD of 1, satisfying the second criterion.

2.4 T - Singularities

2.4.1 Definition: T-singularities

T-singularities are a special type of surface singularity observed in the study of Fano polygons. To understand T-singularities, we need to analyze the edges of a Fano polygon and calculate specific numerical values related to these edges. Let's break down the process and include additional metrics like lattice height and width for a comprehensive view [Reid, 1980; Alexander & Hirschowitz, 1995].

What are T-Singularities?

T-singularities are specific types of singular points on a surface where the surface exhibits a particular form of geometric distortion or complexity. They occur in the study of Fano polygons, which are convex polygons with integer coordinates that contain the origin and have vertices with coordinates that are coprime with the origin. Analyzing T-singularities involves examining the edges of these polygons and calculating certain geometric properties, such as lengths and heights, to understand the nature and quantity of these singularities [Altmann & Ilten, 2012].

Steps to Calculate T-Singularities

1. Understand the Edge of a Polygon:

Each edge of a Fano polygon is a line segment connecting two lattice points. It is imperative to note that in a Fano lattice polygon, the origin (0,0) does not lie on any edge.

2. Calculate the Lattice Length of an Edge:

The lattice length l(E) of an edge E is defined as the number of lattice points on this edge, minus one.

Example: Consider an edge from (1,0) to (4,0). The lattice points on this edge are (1,0), (2,0), (3,0), and (4,0). Thus, the lattice length l(E) = 4 - 1 = 3.

3. Calculate the Lattice Height of an Edge:

The lattice height h(E) of an edge E measures the distance between E and the parallel line segment that passes through the origin. To find h(E):

- Find a Perpendicular Vector: Determine a vector n that is perpendicular to the edge v = (v_x, v_y). A perpendicular vector n can be given by n = ($-v_y$, v_x).
- Normalize the Perpendicular Vector: Normalize n to ensure it has a unit length. The normalized vector n̂ is given by:

$$\hat{\mathbf{n}} = \frac{\mathbf{n}}{\sqrt{v_x^2 + v_y^2}} = \frac{(-v_y, v_x)}{\sqrt{v_x^2 + v_y^2}}$$

- Compute the Height: The lattice height h(E) is the absolute value of the dot product of the normalized perpendicular vector n̂ with the vector representing the origin to one of the vertices of the edge, say v1 = (x_1, y_1). Therefore:

$$h(E) = |\hat{\mathbf{n}} \cdot \mathbf{v_1}| = \left| \frac{-v_y \cdot x_1 + v_x \cdot y_1}{\sqrt{v_x^2 + v_y^2}} \right|$$

4. Calculate the Number of T-Singularities for Each Edge:

The number of T-singularities for an edge E is traditionally defined as the integer ratio l(E)/h(E), where l(E) is the lattice length and h(E) is the lattice height. However, this calculation is valid only when the ratio is an integer. If the ratio is not an integer, it may indicate that the edge does not contribute a straightforward T-singularity, and further geometric or combinatorial analysis might be needed.

Example: Suppose l(E) = 4 and h(E) = 1, then the number of T-singularities is l(E)/h(E) = 4. But if l(E) = 3 and h(E) = 2, then l(E)/h(E) = 1.5, which is not an integer. In such cases, the concept of T-singularity needs to be handled differently, potentially by considering fractional singularities or re-evaluating the lattice polygon's geometric properties.

5. Sum Over All Edges:

To get the total number of T-singularities for the Fano polygon, sum the valid T-singularities calculated for each edge. If any edge does not produce an integer T-singularity, consider whether this contributes a fractional singularity or requires further analysis.

Explicit Example:

Consider a Fano triangle with vertices at (0,1), (1,0), and (-1, -1).

- Edge 1: From (0,1) to (1,0).
  - Lattice length l(E1) = 1 (lattice points: (0,1), (1,0).
  - Perpendicular vector: n1 = (1, -1).
  - Lattice height h(E1) = 1.
  - T-singularities: l(E1)/h(E1) = 1.
- Edge 2: From (1,0) to (-1, -1).
  - Lattice length l(E2) = 1.
  - Perpendicular vector: n2 = (1, -2).
  - Lattice height h(E2) = 1.
  - T-singularities: l(E2)/h(E2) = 1.
- Edge 3: From (-1, -1) to (0,1).
  - Lattice length l(E3) = 1.
  - Perpendicular vector: n3 = (-2,1).
  - Lattice height h(E3) = 1.
  - T-singularities: l(E3)/h(E3) = 1.

2.5 Ehrhart Theory and Its Application

2.5.1 Ehrhart Theory and Fano Polygons

Ehrhart theory is a branch of combinatorial geometry that studies the relationship between the geometry of a convex lattice polytope and the number of lattice points contained within its dilates. For a Fano polygon, the Ehrhart polynomial captures essential geometric information about the polygon. Specifically, the Ehrhart polynomial for 2 dimension is a quadratic polynomial that encodes the number of lattice points inside a scaled version of the polygon.

The Ehrhart polynomial of a Fano polygon can be expressed as:

$$L_P(t) = h_2 \cdot t^2 + h_1 \cdot t + 1$$

Here:

h_1 is the number of lattice points inside the Fano polygon

h_2 is the number of lattice points inside the Fano polygon scaled by factor of 2.

The coefficients of this polynomial provide critical insights into the combinatorial structure of the Fano polygon [Beck & Robins, 2007].

2.5.2 Calculation of h_1 and h_2

To fully understand the Ehrhart polynomial, we need to calculate h_1 and h_2, which are essential for determining the polynomial's coefficients. Here's how you can compute these values:

1. Calculate h_1:

- **Step 1:** Identify and count all the lattice points that lie strictly inside the polygon. These are points with integer coordinates that do not lie on the boundary of the polygon.
- **Example:** For the Fano polygon with vertices at (1,0), (0,1), and (−1, −1), the interior lattice point is (0,0).
- **Count:** The only interior lattice point is (0,0). Thus, h_1=1.

2. Calculate h_2:

- **Step 1:** Scale up the number of lattice points inside Fano Polygon by factor of 2.
- **Example:** For the polygon with vertices (1,0), (0,1), and (−1,−1), list the lattice points on each edge:
  - **Edge from** (1,0) **to** (0,1)**:** Scaled up points are (2,0), (0,2).
  - **Edge from** (0,1) **to** (−1, −1)**:** Scaled up points are (0,2), (-2, -2).
  - **Edge from** (−1, −1) **to** (1,0)**:** Scaled up points are (-2, -2), (2,0).
- **Count:** The lattice points that are strictly inside polygon are (1,0), (0,1), (−1, −1), and (0,0). Thus, h_2=4 (including the interior point (0,0)).

2.6 Summary

In this literature review, we have explored the fundamental concepts necessary for understanding Fano lattice polygons and T-singularities. We have discussed the structural properties of lattice polygons, the specific conditions that define a Fano polygon, and the methods for calculating the number of T-singularities associated with such polygons. By combining combinatorial and algebraic geometry, we have laid the groundwork for further exploration of these mathematical objects, which have significant implications in various areas of geometry and topology.

Chapter 3: Literature Review - Machine Learning Applications

3.1 Introduction

In recent years, the intersection of machine learning and algebraic geometry has emerged as a promising frontier for advancing both theoretical and computational aspects of these fields. This chapter dives into the application of machine learning techniques to the study of Fano lattice polygons, with a particular focus on predicting T-singularities—a critical challenge in algebraic geometry. The use of machine learning in this context represents a significant shift from traditional methods, which have historically relied on combinatorial and geometric approaches. These traditional methods, while powerful, often struggle with the computational complexity and the high-dimensional nature of the problems posed by Fano lattice polygons and their associated singularities.

The primary objective of this chapter is to provide a comprehensive review of machine learning models that can be used to enhance the prediction of number of T-singularities within Fano lattice polygons. The models explored include Logistic Regression, Random Forests, Support Vector Classification (SVC), and Gradient Boosting, each of which offers unique advantages in terms of accuracy, interpretability, and computational efficiency. The integration of these models into the study of Fano lattice polygons is not merely a matter of applying established techniques to a new domain; it involves a careful adaptation of machine learning methods to accommodate the specific characteristics of algebraic data.

Logistic Regression, as a foundational statistical model, is explored for its ability to provide interpretable results and identify linear relationships between the Ehrhart coefficients of Fano polygons and the presence of T-singularities. Random Forests and Gradient Boosting, as ensemble learning methods, are reviewed for their capacity to model complex, non-linear relationships and to handle the high-dimensional datasets typical of algebraic geometry. Support Vector Classification (SVC), with its robust handling of high-dimensional spaces and its use of kernel functions to manage non-linearity, is also evaluated for its suitability in this context.

Beyond the selection and application of machine learning models, this chapter addresses the critical role of feature engineering in the successful prediction of number of T-singularities. Feature engineering involves creating new features from existing data that capture the underlying structure and relationships within Fano polygons. Techniques such as Polynomial Features and Principal Component Analysis (PCA) are discussed in detail, with an emphasis on how they can be used to enhance model performance by revealing hidden patterns in the data.

Furthermore, the chapter highlights the importance of evaluation metrics in assessing the effectiveness of machine learning models in this domain. Metrics such as accuracy, precision, recall, and F1-score are not merely statistical measures; they are tools that provide insight into the model's ability to generalize from training data to unseen examples. These metrics are particularly important in the context of T-singularity prediction, where the consequences of false positives and false negatives can have significant implications for the understanding of algebraic structures.

The integration of machine learning into the study of Fano lattice polygons is a novel approach that has the potential to significantly advance our understanding of T-singularities. By understanding the power of machine learning, this research aims to overcome the limitations of traditional methods, offering new tools and techniques for algebraic geometers. The following sections of this chapter provide a detailed review of the relevant machine learning models, feature engineering strategies, and evaluation metrics, laying the groundwork for their application to the mathematical data collected in this study. Through this review, we aim to establish a strong foundation for the innovative use of machine learning in algebraic geometry, particularly in the challenging task of predicting number of T-singularities within Fano lattice polygons.

3.2Machine Learning in Algebraic Geometry

3.2.1 Introduction to Machine Learning Applications

Machine learning, with its capability to model non-linear relationships, has become an essential tool in various scientific disciplines, including algebraic geometry. It provides new methods for classifying algebraic varieties, predicting geometric properties, and analyzing combinatorial structures (Bishop, 2006).

3.2.2 Predicting T-singularities Using Machine Learning

Machine learning can predict T-singularities in Fano lattice polygons by using combinatorial data. Models trained on datasets of known polygons and singularities can offer more accurate and generalizable predictions than traditional methods (Montufar, Pascanu, Cho, & Bengio, 2014).

3.2.3 Existing Studies and Techniques

Several studies have shown the effectiveness of machine learning in algebraic geometry. Techniques such as neural networks, support vector machines, and random forests have been used to identify patterns in combinatorial data, providing new insights into the structure of algebraic varieties (Hastie, Tibshirani, & Friedman, 2009).

Table 1: Comparison of Traditional and Machine Learning Methods for Predicting T-singularities. This table contrasts different approaches to predicting singularities, highlighting their advantages and limitations.

Aspect	**Traditional Methods**	**Machine Learning Methods**
Approach	Analytical and algebraic techniques, based on theorems and formulas.	Data-driven methods, using training data to build predictive models.
Complexity	Can be complex and require deep mathematical knowledge.	Depends on the model used; some models require significant computational resources.
Flexibility	Limited to specific cases that can be handled analytically.	High flexibility; models can adapt to various types of data.
Accuracy	High for well-understood problems with clear mathematical solutions.	Potentially higher with sufficient data but depends on model quality.
Interpretability	Results are often easy to interpret in mathematical terms.	Can be difficult to interpret, especially with complex models like deep learning.
Scalability	May not scale well with large or complex datasets.	Highly scalable with large datasets and parallel processing.
Application Areas	Best suited for problems where the mathematical structure is well known.	Applicable to a wide range of problems, including those without clear mathematical models.

3.3 Model Selection

Model selection is a critical step in any machine learning project, as it involves choosing the most appropriate algorithms that align with the specific problem, dataset characteristics, and analysis goals. The selection process is essential for achieving high-performance models that perform well on unseen data. In the context of this study, various machine learning models were considered to predict T-singularities in Fano lattice polygons. This section provides an in-depth review of each selected model, including their theoretical foundations, mathematical formulations, suitability for the task, and advantages. In this project, several models were evaluated to identify the best approach for the given classification problem (Breiman, 2001; Friedman, 2001).

Choice of Models

1. **Logistic Regression:**

   - **Purpose:** Logistic Regression is a statistical model widely used for binary classification tasks, where the goal is to predict the probability of a binary outcome based on one or more predictor variables. It is particularly useful when the relationship between the predictors and the outcome is approximately linear. The model estimates the probability of the positive class, which can be interpreted as the likelihood of a particular event occurring. (Hosmer, Lemeshow, & Sturdivant, 2013).

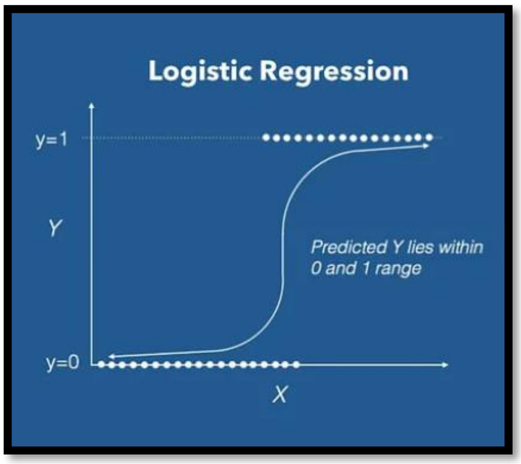


Fig5: Logistic Regression Graphical Representation

- **Mathematical Formula:**

$$\boxed{\mathrm{Logit}(p) = \log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n}$$

$$\boxed{p = \frac{1}{1+e^{-(B_0+B_1X_1+B_2X_2+\cdots+B_nX_n)}}}$$

- **Where:**
  - p is probability of positive class.
  - $X_1, X_2, \ldots X_n$ are predictor variables.
  - $B_0, B_1, \ldots B_n$ are the coefficients estimated by model.
- **Components of the Formula:**
  - ***Linear Combination (Logit):***
    - The term inside the exponent, is a linear combination of predictor variables. This is often referred to as the "logit" or "log-odds" of the positive class.
    - B_0 is the intercept term (constant).
    - $B_1, \ldots B_n$ are the coefficients that represent the influence of each predictor variable $X_1, X_2, \ldots X_n$ on the probability.
  - **Exponential Function:**
    - The term $e^{-(B_0+B_1X_1+B_2X_2+\cdots+B_nX_n)}$ represents the exponential of the negative logit. This is used to ensure that the output probability p is between 0 and 1.
  - **Probability Calculation:**
    - The formula $p = \frac{1}{1+e^{-(B_0+B_1X_1+B_2X_2+\cdots+B_nX_n)}}$ uses the sigmoid function to transform the linear combination of predictors into a probability. The sigmoid function maps any real-valued number into the range (0, 1).

- **Suitability:** This model is particularly suitable for problems where the relationship between the predictors and the binary outcome is approximately linear. It is favored for its simplicity, interpretability, and efficiency in handling large datasets (Cortes & Vapnik, 1995).

- **Advantages:**

  Interpretability: The coefficients in Logistic Regression provide direct insights into the influence of each predictor variable on the outcome.

  Efficiency: It is computationally efficient and can handle large datasets effectively.

  Simple Implementation: Logistic Regression is easy to implement using standard libraries and tools.

2. Random Forest:

- **Purpose:** Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the class that is the majority vote from all trees (Breiman, 2001).

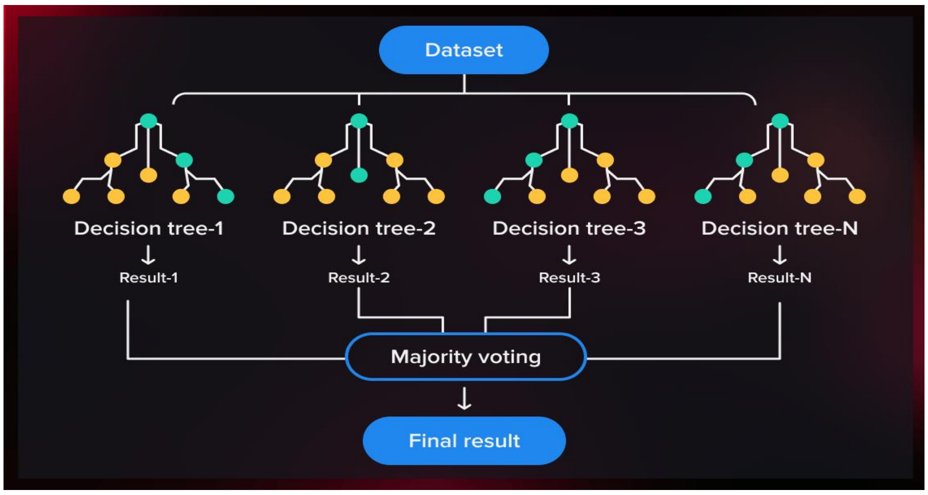


Fig 6: Random Forest Graphical Representation

- **Mathematical Formula for Classification:**

$$\boxed{\hat{y} = \text{mode}\left(T_1(X), T_2(X), \ldots, T_K(X)\right)}$$

- **Where:**
  - **ŷ**: This represents the final predicted class label for the input X.
  - ***$T_k(X)$***: Each $T_k(X)$ is the prediction from the k-th decision tree in the forest for the input X. Since this is a classification task, each tree outputs a class label (e.g., "yes" or "no", "spam" or "not spam").
  - mode (·): The mode function takes all the class labels predicted by the K trees and returns the most frequent class label among them. For example, if 3 out of 5 trees predict "yes" and 2 trees predict "no," the mode is "yes."
  - K: This is the total number of decision trees in the Random Forest.
- **Mathematical Formula for Regression:**

$$\boxed{\hat{y} = \frac{1}{K}\sum_{k=1}^{K} T_k(X)}$$

- **Where:**
  - $\hat{y}$: This represents the final predicted class label for the input X.
  - *T*$_k$*(X)*: Each T$_k$(X) is the prediction from the k-th decision tree in the forest for the input X. In regression, each tree outputs a numerical value rather than a class label.
  - $\frac{1}{k}\sum_{k=1}^{k} Tk\ (x)$This expression calculates the average of the predictions from all K trees.

- **Suitability:** This model is well-suited for capturing complex relationships and interactions between features due to its ensemble nature. It handles non-linearities and interactions between features effectively (Liaw & Wiener, 2002).
- **Advantages:**

  Reduces Overfitting: By averaging the results of multiple trees, Random Forests reduce the risk of overfitting.

  Feature Importance: Random Forests provide metrics that indicate the importance of different features, making it easier to understand the significance of each variable.

  Handles Missing Data: Random Forests can handle missing data effectively by using imputed values.

3. **Gradient Boosting:**

- **Purpose:** Gradient Boosting is another ensemble technique that builds models sequentially, where each model tries to correct the errors of its predecessor. It combines multiple weak learners (typically decision trees) to create a robust predictive model (Friedman, 2001).

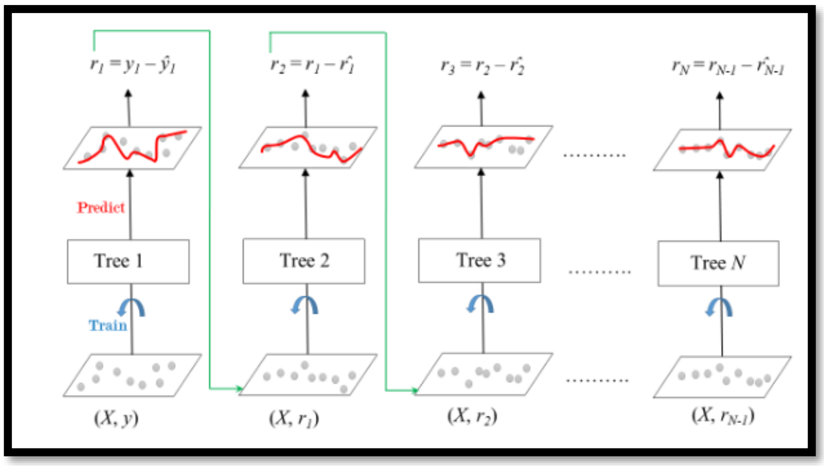


Fig 7 : Gradient Boosting Architecture

- **Mathematical Formula:**

$$\boxed{F_m(x) = F_{m-1}(x) + \nu \cdot h_m(x)}$$

- **Where:**
  - ➢ *$F_m(x)$* is the model at the m-th iteration.
  - ➢ *$h_m(x)$* is the weak learner added at the m-th iteration.
  - ➢ *v* is the learning rate.
- **Components of the Formula:**

Initial Model *$F_0(x)$*: The process starts with an initial model *$F_0(x)$*, which could be a simple guess such as the mean of the target values in regression or the log odds in classification.

Updating the Model:

$F_{m-1}(x)$: This is the prediction of the model after m−1 iterations. It represents the accumulated predictions from all previously added weak learners.

$h_m(x)$: This is the new weak learner added in the m-th iteration. It is trained to predict the residuals (errors) of the current model *$F_{m-1}(x)$*. Essentially, *$h_m(x)$* tries to capture the errors or shortcomings of the model so far.

Learning Rate ν. *$h_m(x)$*: ν. *$h_m(x)$* (often called the shrinkage parameter) scales the contribution of the new weak learner. A smaller learning rate means that each weak learner has a smaller impact on the final model, which usually requires more iterations to converge but can lead to a more robust model.

Updated Model $F_m(x)$: The new prediction $F_m(x)$ is obtained by adding the scaled prediction of the new weak learner $\nu \cdot h_m(x)$ to the previous model's prediction $F_{m-1}(x)$.This update process aims to correct the errors made by the previous iteration, thereby improving the model incrementally.

- **Suitability:** This model is effective in improving predictive performance, especially when dealing with complex datasets with non-linear relationships (Friedman, 2001).
- **Advantages:**

  High Accuracy: Gradient Boosting often achieves better accuracy compared to individual models by reducing bias.

  Focus on Hard Cases: The model emphasizes difficult cases, which improves overall prediction performance.

  Flexibility: Gradient Boosting can be adapted to various types of loss functions, making it versatile for different tasks.

4. **Support Vector Machines (SVM):**

- **Purpose:** SVM is a powerful classification algorithm that finds the hyperplane that best separates classes in the feature space. It is effective in high-dimensional spaces and for cases where the boundary between classes is not linear (Cortes & Vapnik, 1995).

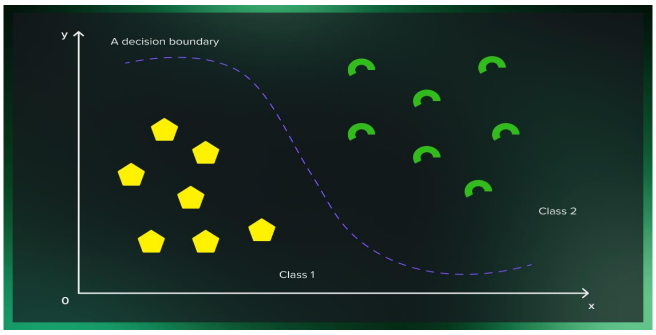


Fig 8: Support Vector Machine Graphical Representation

Mathematical Formula:

$$\boxed{\text{Decision Function: } f(x) = \text{sign}\left(\sum_{i=1}^{N} \alpha_i y_i K(x_i, x) + b\right)}$$

Where:

- α_i are the Lagrange multipliers.
- y_i is the class label.
- $K(x_i, x)$ is the kernel function.
- b is the bias term.

- **Suitability:** This model is ideal for problems where the classes are not easily separable and can handle high-dimensional data effectively (Vapnik, 1998).
- **Advantages:**

• **Handles non-linearity:** SVMs can handle non-linear data effectively using different kernel functions.

• **Robustness:** SVMs are less prone to overfitting, making them suitable for complex classification tasks.

• **Flexibility:** The ability to use various kernel functions allows SVMs to adapt to a wide range of data structures.

5. **K-Means Clustering:**

- **Purpose:** K-Means is an unsupervised learning algorithm used for clustering data into a predefined number of clusters. It helps in identifying patterns and grouping similar data points (MacQueen, 1967).

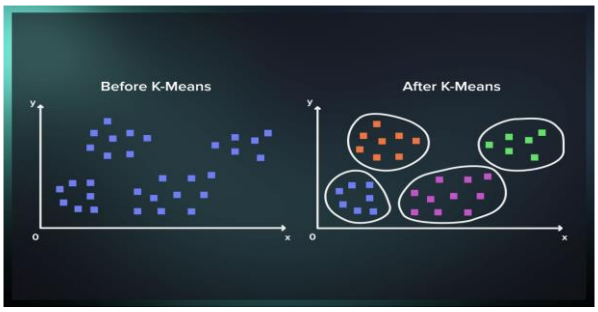


Fig 9: K-Means cluster representation

- **Mathematical Formula:**

$$\boxed{\text{Minimize} \sum_{i=1}^{k} \sum_{x \in C_i} ||x - \mu_i||^2}$$

- **Where:**
  - k is the number of clusters.
  - C_i is the ith cluster.
  - μ_i is the centroid of ith cluster.

- **Suitability:** Although primarily used for unsupervised learning, K-Means was included for baseline comparison. It can provide insights into the natural grouping within the data (MacQueen, 1967).

- **Advantages:**

  • **Simplicity:** K-Means is easy to implement and understand, making it accessible for various applications.

  • **Efficiency:** The algorithm is computationally efficient, especially for large datasets.

  • **Pattern Recognition:** K-Means is effective at identifying natural groupings within data, which can be useful for exploratory analysis.

3.4 Feature Engineering and Dimensionality Reduction

Feature engineering is a important step in the application of machine learning to any problem, including the analysis of Fano lattice polygons. This process involves creating new features from the original data that can enhance the predictive power of the model. In the context of Ehrhart theory and T-singularities, feature engineering could involve creating polynomial features, interaction terms, or other derived features that capture the complex relationships between the coefficients of the Ehrhart polynomial and the number of T-singularities (Bishop, 2006).

Dimensionality reduction is another important technique in machine learning, particularly when dealing with high-dimensional data. Techniques such as Principal Component Analysis (PCA) can reduce the dimensionality of the data by identifying the most important features or combinations of features that capture the majority of the variance in the data. In the study of Fano lattice polygons, dimensionality reduction could be used to simplify the data, making it easier to identify patterns and make predictions about T-singularities (Jolliffe, 2011).

The combination of feature engineering and dimensionality reduction has the potential to significantly improve the performance of machine learning models in predicting T-singularities. By carefully selecting and creating features, and by reducing the dimensionality of the data, it is possible to develop more accurate and interpretable models. This approach is expected to be a key component of the methodology used in this dissertation (Hastie, Tibshirani, & Friedman, 2009).

3.5 Model Validation

Model validation is a important step in the machine learning pipeline that ensures models perform well on unseen data and generalize effectively. A fundamental component of model validation is cross-validation, a method used to evaluate a model's performance by partitioning the data into subsets. This helps in assessing how good the model generalizes to new data and in picking the best model and tuning its hyperparameters (Hastie, Tibshirani, & Friedman, 2009).

1. Importance of Model Validation

Model validation aims to avoid overfitting and underfitting by providing an estimate of the model's generalization ability. Effective model validation helps in understanding the reliability and robustness of the model in practical scenarios (James, Witten, Hastie, & Tibshirani, 2013). By employing various cross-validation techniques, one can ensure that the model performs consistently across different subsets of data.

2. Cross-Validation Techniques

Cross-validation involves several methods, each suitable for different types of data and modeling tasks. The main techniques include:

- 2.1. K-Fold Cross-Validation: K-Fold Cross-Validation divides the dataset into k equally sized folds. The model gets trained on k−1 of these folds and validated on the remaining fold. This process is repeated k times, with each fold serving as the validation set once. The performance metrics are averaged over all folds to provide a comprehensive evaluation of the model (Kohavi, 1995). This technique is widely used due to its ability to provide a reliable estimate of model performance by ensuring that every data point is used for both training and validation.
- 2.2. Leave-One-Out Cross-Validation (LOOCV): LOOCV is a special case of k-fold cross-validation where k is equal to the number of data points in the dataset. In this method, each data point is used as a validation set exactly once, while the remaining data points are used for training. This approach gives an almost unbiased estimate of model performance but can be computationally intensive for large datasets (Kohavi, 1995).
- 2.3. Stratified Cross-Validation: Stratified Cross-Validation is used in classification problems where it is important to maintain the same proportion of class labels in each fold as in the original dataset. This technique helps in providing a more accurate estimate of model performance, particularly when dealing with imbalanced classes (Elisseeff & Weston, 2002). By preserving class distribution, stratified cross-validation ensures that each fold is representative of the overall data.

- 2.4. Time-Series Cross-Validation: Time-Series Cross-Validation is adapted for time-series data where maintaining the temporal order is important. Methods such as rolling-window and expanding-window approaches are used to split the data while respecting its sequential nature. Rolling-window cross-validation involves training on a fixed-size window of data that moves forward in time, whereas expanding-window cross-validation uses an increasing training set as time progresses (Hyndman & Athanasopoulos, 2018). This technique helps in evaluating models in forecasting and other time-dependent tasks.

3. Practical Considerations

Choosing the right cross-validation technique depends on the nature of the data and the specific problem being addressed. For instance, stratified cross-validation is ideal for classification tasks with imbalanced classes, while time-series cross-validation is necessary for data with temporal dependencies. Additionally, cross-validation can be computationally intensive, so balancing computational resources with the need for robust evaluation is essential.

3.6 Performance Metrics

In the realm of machine learning models, particularly those designed for detecting and analyzing number of T-singularities, performance metrics are indispensable for evaluating the accuracy, robustness, and overall effectiveness of the models. These metrics play a critical role in determining how well a model performs in capturing and quantifying complex geometrical properties within lattice polygons. The selection of appropriate metrics can profoundly impact the assessment and refinement of these models. This section dives into the key performance metrics employed to evaluate various dimensions of model performance in the context of T-singularity detection.

1. **Accuracy**: Accuracy is the ratio of correctly predicted observations to the total observations. It is a commonly used metric for evaluating classification models. Accuracy gives the overall effectiveness of a model but can be misleading if the data is imbalanced.

$$\boxed{Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}}$$

2. **Recall**: Recall, also known as Sensitivity or True Positive Rate, is the ratio of correctly predicted positive observations to all observations in the actual class. It is important when the focus is on capturing all the relevant cases (e.g., in medical diagnostics where missing a positive case can be critical).

$$\boxed{Precision\ Score = \frac{True\ Positives}{True\ Positives\ +\ False\ Positives}}$$

3. **Precision Score**: Precision, also known as the Positive Predictive Value, is the ratio of correctly predicted positive observations to the total predicted positive observations. It focuses on the accuracy of positive predictions, making it useful in scenarios where the cost of false positives is high.

$$\boxed{Recall = \frac{True\ Positives}{True\ Positives\ +\ False\ Negatives}}$$

3.7 Recent Advances and Future Directions

Recent advances in machine learning, particularly in deep learning and ensemble methods, have opened new avenues for research in algebraic geometry. These techniques offer the potential to model even more complex relationships and to make more accurate predictions than traditional methods.

Deep learning, with its ability to learn hierarchical representations of data, is particularly well-suited to problems involving high-dimensional data, such as the analysis of Ehrhart polynomials and T-singularities (LeCun, Bengio, & Hinton, 2015).

Ensemble methods, which combine the predictions of multiple models, have also shown promise in improving the accuracy and robustness of machine learning models. Techniques such as random forests, gradient boosting, and stacking have been successfully applied to a wide range of problems and are expected to play a key role in the future development of predictive models in algebraic geometry (Breiman, 2001).

Looking forward, the integration of machine learning with traditional combinatorial and geometric methods is likely to yield new insights and tools for the study of Fano lattice polygons and T-singularities. By understanding the strengths of both approaches, researchers can develop more powerful and generalizable models, opening new directions for research in algebraic geometry (Montufar et al., 2014).

3.8 Summary

This chapter reviewed the integration of machine learning with algebraic geometry, focusing on predicting T-singularities in Fano lattice polygons. It covered various machine learning models, including Logistic Regression, Random Forests, Gradient Boosting, and SVMs, highlighting their applications and advantages. Key techniques such as feature engineering, dimensionality reduction, and cross-validation were discussed to enhance model performance and validation. Recent advancements in deep learning and ensemble methods were noted for their potential to further improve predictive accuracy. Overall, the chapter establishes a foundation for using machine learning to address complex problems in algebraic geometry.

Chapter 4: Methodology

4.1 Research Design

The study of lattice polygons and their associated singularities has traditionally relied on geometric and combinatorial techniques to analyze Tsingularities (Bishop, 2006; Jolliffe, 2002). However, these conventional methods often face limitations in terms of scalability and efficiency, particularly when dealing with complex or high-dimensional cases (Murphy, 2012).

Our research introduces a novel approach by focusing exclusively on the number of Tsingularities within lattice polygons. This specialization allows for a more streamlined and precise analysis compared to traditional methods that may encompass a broader range of singularities. By integrating advanced machine learning techniques into our approach (Bergstra & Bengio, 2012), we aim to address the challenges of efficiency and scalability.

4.1.1 Goal

The primary goal of this research is to precisely quantify and analyze the number of T-singularities in lattice polygons through the application of machine learning methodologies (Breiman, 2001). By focusing on the number of Tsingularities, we seek to gain a detailed understanding of their distribution and characteristics, contributing to the broader classification and study of Fano varieties (Cortes & Vapnik, 1995). Our approach aims to improve the efficiency and scalability of singularity analysis, enabling the exploration of more complex and high-dimensional scenarios than conventional methods permit.

4.1.2 Objective

The specific objectives of our research are:

- **Precision:** To deliver an accurate count and detailed characterization of the number of Tsingularities in lattice polygons (Friedman, 2001).
- **Efficiency:** To utilize machine learning algorithms to accelerate the analysis process and mitigate the computational challenges faced by traditional methods (Kohavi, 1995).
- **Scalability:** To enhance the capability to analyze more intricate and higher-dimensional lattice polygons, providing a comprehensive understanding of T-singularities across a broader range of examples (Little & Rubin, 2019).

4.1.3 Research Architecture:

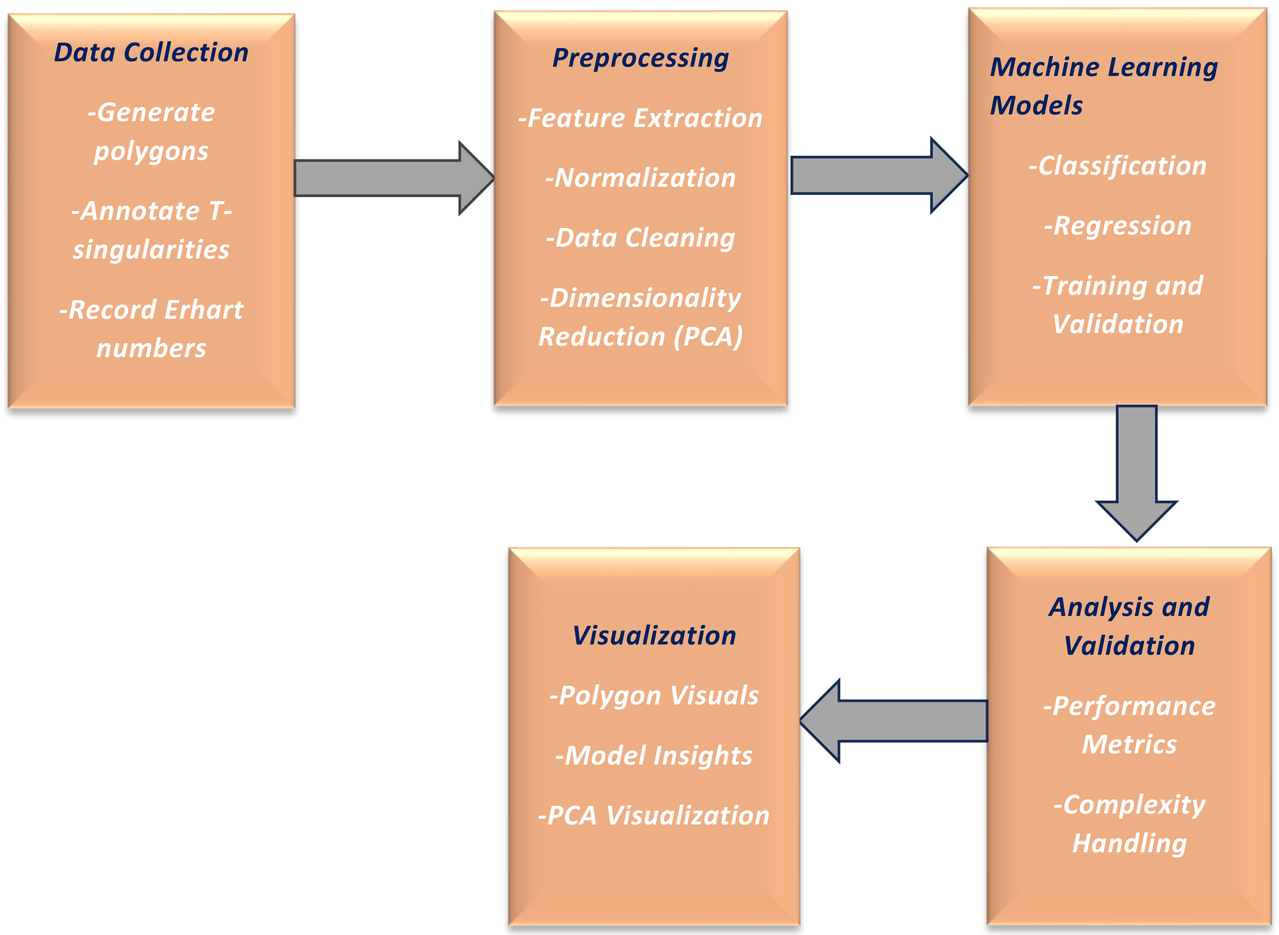


Fig 10: Architecture Diagram for Research Design

4.2 Data Collection

The data collection process is foundational to our research methodology, providing the essential data required for analyzing lattice polygons and their associated characteristics, such as T-singularities and Ehrhart numbers (McKinney, 2010).

1. Generation of Lattice Polygons: We generate a large number of random lattice polygons using specialized algorithms that ensure diversity in shape and structure (Lloyd, 1982). These polygons are defined by vertices with integer coordinates, ensuring they lie on a lattice. The number of vertices is carefully selected to create polygons of varying complexity, from simple triangles to more intricate shapes. Each polygon is verified to ensure it meets specific geometric criteria, such as convexity and inclusion of the origin (Bishop, 2006).

Grid-Based Polygon Generation Process

1. Random Point Generation:

- We generate random points on a grid with integer coordinates within a specified range (e.g., -50 to 50).
- Points are selected only if their coordinates are coprime, meaning the greatest common divisor of the coordinates is 1. This helps ensure diversity in the generated points.

2. Bias in Point Selection:

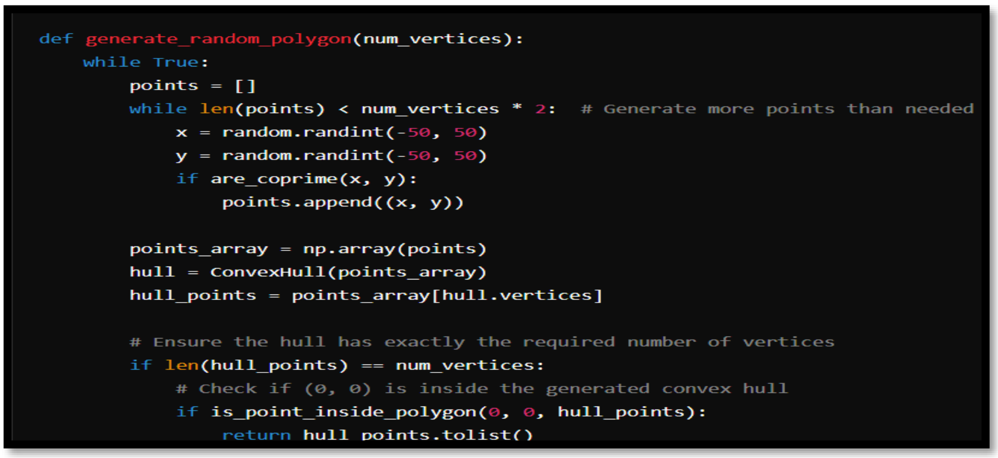
- The choice of the grid range (e.g., -50 to 50) introduces a potential bias. By restricting points within this limited range, the dataset may not fully represent the entire space of possible lattice points, which could affect the generalizability of the findings. This limitation should be considered when interpreting the results and assessing the model's performance.

3. Convex Hull Computation:

- A convex hull is computed from the generated points using the ConvexHull function from the scipy.spatial module. This ensures that the polygons are simple and convex.

4. Validation:

- We check if the convex hull has the required number of vertices. If not, additional points are generated until a suitable polygon is obtained.
- We verify that the origin (0,0) is inside the generated convex hull to ensure the polygon meets the inclusion criterion.



```python
def generate_random_polygon(num_vertices):
    while True:
        points = []
        while len(points) < num_vertices * 2:  # Generate more points than needed
            x = random.randint(-50, 50)
            y = random.randint(-50, 50)
            if are_coprime(x, y):
                points.append((x, y))

        points_array = np.array(points)
        hull = ConvexHull(points_array)
        hull_points = points_array[hull.vertices]

        # Ensure the hull has exactly the required number of vertices
        if len(hull_points) == num_vertices:
            # Check if (0, 0) is inside the generated convex hull
            if is_point_inside_polygon(0, 0, hull_points):
                return hull_points.tolist()
```

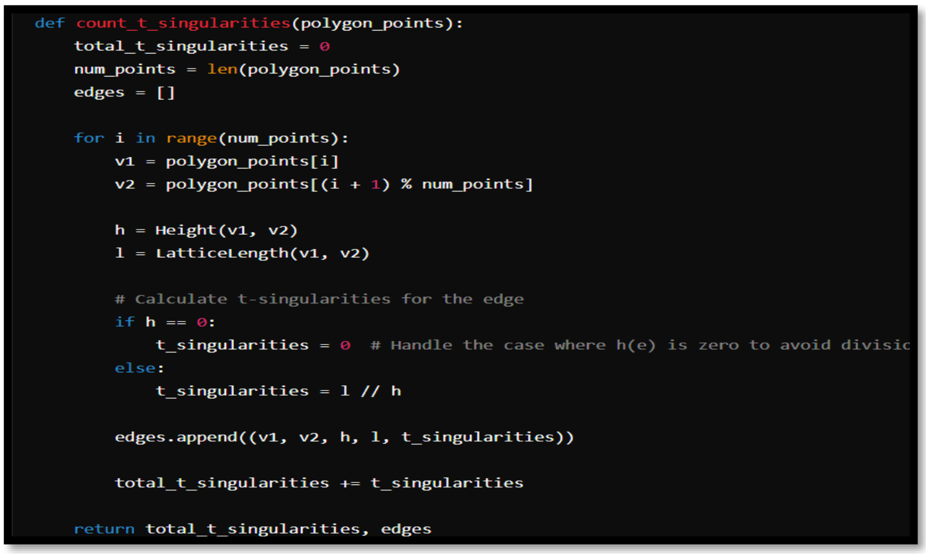
Fig 11: Code for Generating Random Polygons

2. Annotation with T-Singularities: For each lattice polygon, we compute the number of T-singularities, which are critical points where the polygon's Ehrhart quasi-polynomial changes

(Hosmer, Lemeshow, & Sturdivant, 2013). This calculation involves determining the height and lattice length of each edge of the polygon and using these values to compute the T-singularities. This step is automated to ensure consistent and accurate results across the dataset.

T-Singularities Calculation:

- For each edge of the polygon, compute the height and lattice length.
- The number of T-singularities for each edge is calculated and summed to provide the total count for the polygon.

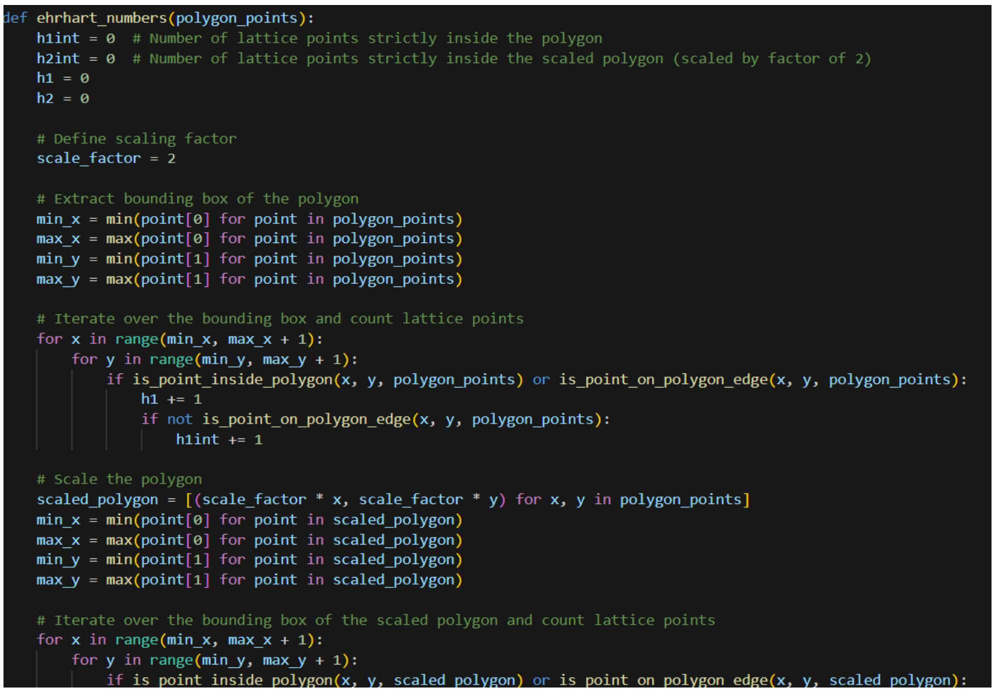


```python
def count_t_singularities(polygon_points):
    total_t_singularities = 0
    num_points = len(polygon_points)
    edges = []

    for i in range(num_points):
        v1 = polygon_points[i]
        v2 = polygon_points[(i + 1) % num_points]

        h = Height(v1, v2)
        l = LatticeLength(v1, v2)

        # Calculate t-singularities for the edge
        if h == 0:
            t_singularities = 0  # Handle the case where h(e) is zero to avoid divisic
        else:
            t_singularities = l // h

        edges.append((v1, v2, h, l, t_singularities))

        total_t_singularities += t_singularities

    return total_t_singularities, edges
```

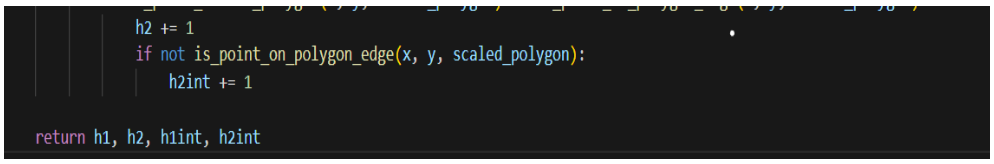
Fig 12: Code for calculating number of T- singularities

3. Calculation of Ehrhart Numbers and Features: Alongside the T-singularities, we compute key Ehrhart-related features for each polygon:

   - h_1: The total number of lattice points within the polygon (Fawcett, 2006).
   - h_2: The number of lattice points within a scaled version of the polygon.
   - h_1 int: The number of lattice points strictly inside the polygon, excluding those on the edges.
   - h_2 int: The number of lattice points strictly inside the scaled polygon, excluding edge points.



```python
def ehrhart_numbers(polygon_points):
    h1int = 0  # Number of lattice points strictly inside the polygon
    h2int = 0  # Number of lattice points strictly inside the scaled polygon (scaled by factor of 2)
    h1 = 0
    h2 = 0

    # Define scaling factor
    scale_factor = 2

    # Extract bounding box of the polygon
    min_x = min(point[0] for point in polygon_points)
    max_x = max(point[0] for point in polygon_points)
    min_y = min(point[1] for point in polygon_points)
    max_y = max(point[1] for point in polygon_points)

    # Iterate over the bounding box and count lattice points
    for x in range(min_x, max_x + 1):
        for y in range(min_y, max_y + 1):
            if is_point_inside_polygon(x, y, polygon_points) or is_point_on_polygon_edge(x, y, polygon_points):
                h1 += 1
                if not is_point_on_polygon_edge(x, y, polygon_points):
                    h1int += 1

    # Scale the polygon
    scaled_polygon = [(scale_factor * x, scale_factor * y) for x, y in polygon_points]
    min_x = min(point[0] for point in scaled_polygon)
    max_x = max(point[0] for point in scaled_polygon)
    min_y = min(point[1] for point in scaled_polygon)
    max_y = max(point[1] for point in scaled_polygon)

    # Iterate over the bounding box of the scaled polygon and count lattice points
    for x in range(min_x, max_x + 1):
        for y in range(min_y, max_y + 1):
            if is_point_inside_polygon(x, y, scaled_polygon) or is_point_on_polygon_edge(x, y, scaled_polygon):
```

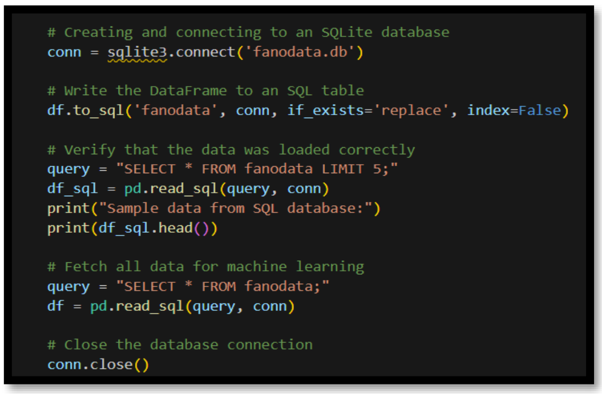


```python
                h2 += 1
                if not is_point_on_polygon_edge(x, y, scaled_polygon):
                    h2int += 1

    return h1, h2, h1int, h2int
```

Fig 13: Code for calculating Erhart numbers

These features provide a comprehensive understanding of the polygon's lattice structure, which is important for analyzing the relationship between lattice geometry and T-singularities (McKinney, 2010).

4. Data Storage and Management: All generated polygons, along with their associated T-singularities, Ehrhart numbers, and additional features, are systematically stored in a structured dataset. Each entry contains detailed information about the polygon's vertices, T-singularities, and Ehrhart-related features. This dataset is carefully managed to ensure it is accessible and ready for analysis in subsequent research phases (SQLite Documentation, n.d.).



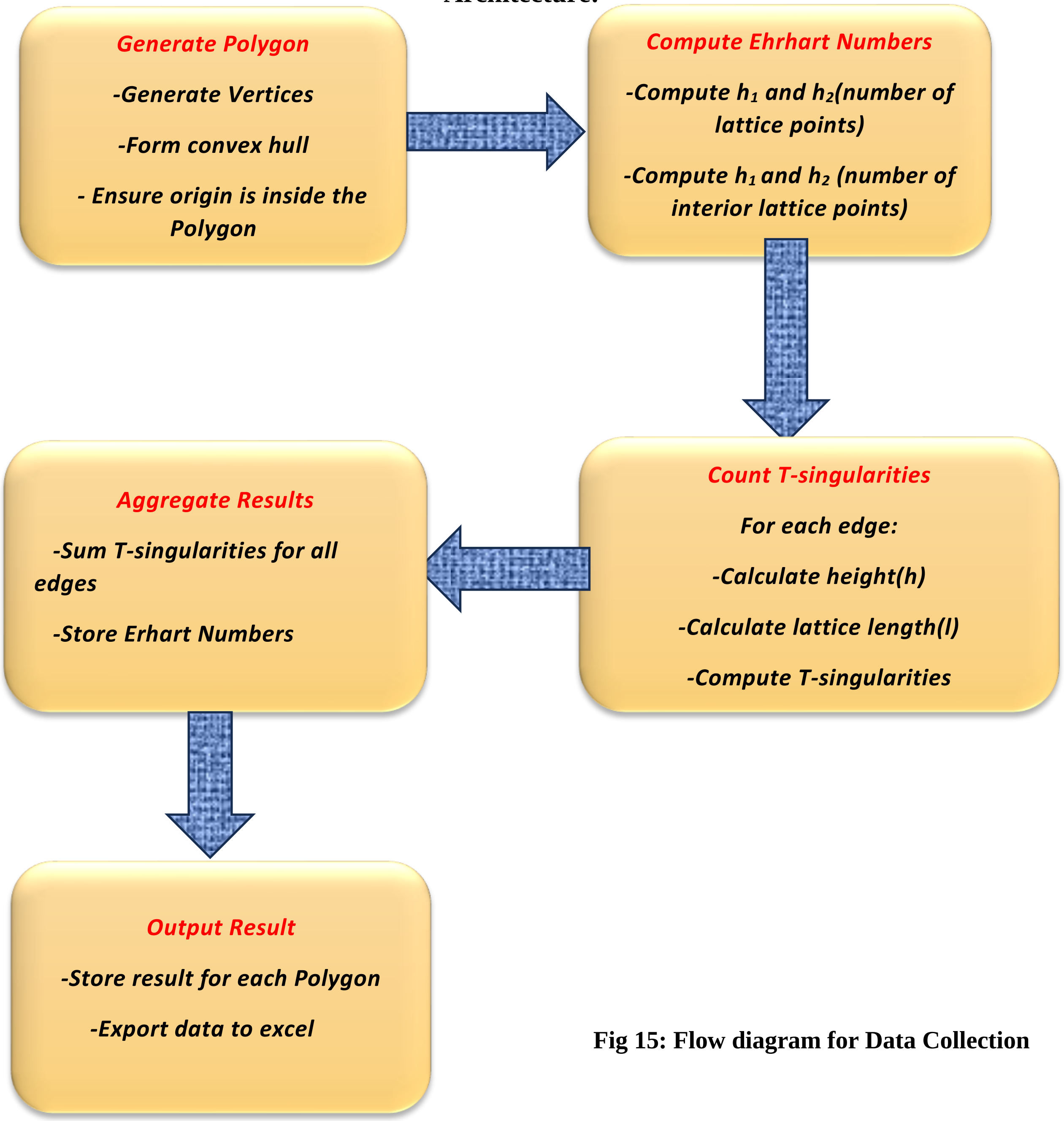
```python
# Creating and connecting to an SQLite database
conn = sqlite3.connect('fanodata.db')

# Write the DataFrame to an SQL table
df.to_sql('fanodata', conn, if_exists='replace', index=False)

# Verify that the data was loaded correctly
query = "SELECT * FROM fanodata LIMIT 5;"
df_sql = pd.read_sql(query, conn)
print("Sample data from SQL database:")
print(df_sql.head())

# Fetch all data for machine learning
query = "SELECT * FROM fanodata;"
df = pd.read_sql(query, conn)

# Close the database connection
conn.close()
```

Fig 14: Code for Retrieving data from SQL

5. Data Augmentation: To enhance the dataset's diversity, we apply data augmentation techniques, such as rotation and scaling of polygons, generating additional data points. This augmentation allows us to capture a broader range of geometric configurations and improves the robustness of our computational models (Pandas Documentation, n.d.). Each augmented polygon is added as a new entry in the dataset, with T-singularities and Ehrhart features recalculated.

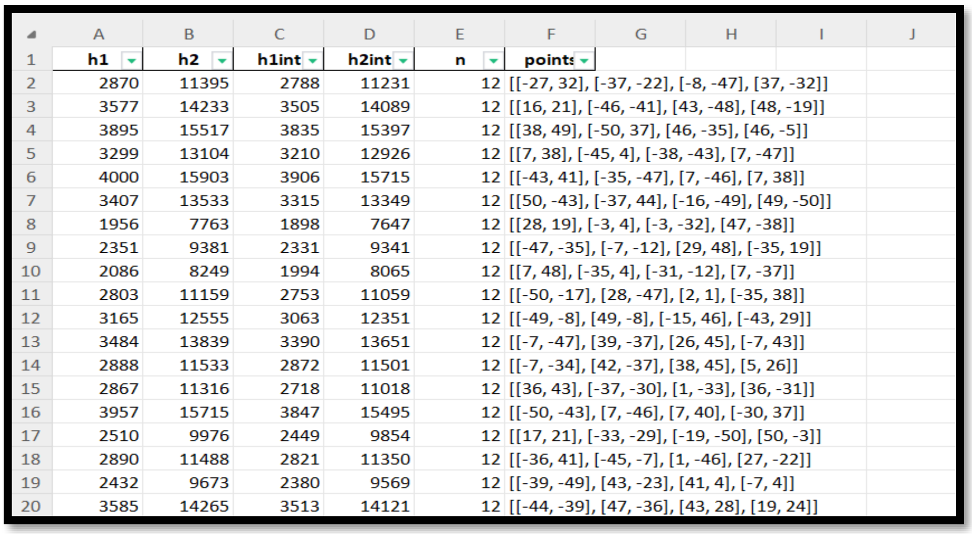
This extensive dataset, enriched with T-singularities and Ehrhart-related features, serves as the foundation for training and validating our models, enabling a deeper exploration of the relationship between lattice geometry and T-singularities.

Architecture:



Generate Polygon
-Generate Vertices
-Form convex hull
- Ensure origin is inside the Polygon
Compute Ehrhart Numbers
-Compute h_1 and h_2(number of lattice points)
-Compute h_1 and h_2 (number of interior lattice points)
Count T-singularities
For each edge:
-Calculate height(h)
-Calculate lattice length(l)
-Compute T-singularities
Aggregate Results
-Sum T-singularities for all edges
-Store Erhart Numbers
Output Result
-Store result for each Polygon
-Export data to excel

Fig 15: Flow diagram for Data Collection



	A	B	C	D	E	F
1	h1	h2	h1int	h2int	n	points
2	2870	11395	2788	11231	12	[[-27, 32], [-37, -22], [-8, -47], [37, -32]]
3	3577	14233	3505	14089	12	[[16, 21], [-46, -41], [43, -48], [48, -19]]
4	3895	15517	3835	15397	12	[[38, 49], [-50, 37], [46, -35], [46, -5]]
5	3299	13104	3210	12926	12	[[7, 38], [-45, 4], [-38, -43], [7, -47]]
6	4000	15903	3906	15715	12	[[-43, 41], [-35, -47], [7, -46], [7, 38]]
7	3407	13533	3315	13349	12	[[50, -43], [-37, 44], [-16, -49], [49, -50]]
8	1956	7763	1898	7647	12	[[28, 19], [-3, 4], [-3, -32], [47, -38]]
9	2351	9381	2331	9341	12	[[-47, -35], [-7, -12], [29, 48], [-35, 19]]
10	2086	8249	1994	8065	12	[[7, 48], [-35, 4], [-31, -12], [7, -37]]
11	2803	11159	2753	11059	12	[[-50, -17], [28, -47], [2, 1], [-35, 38]]
12	3165	12555	3063	12351	12	[[-49, -8], [49, -8], [-15, 46], [-43, 29]]
13	3484	13839	3390	13651	12	[[-7, -47], [39, -37], [26, 45], [-7, 43]]
14	2888	11533	2872	11501	12	[[-7, -34], [42, -37], [38, 45], [5, 26]]
15	2867	11316	2718	11018	12	[[36, 43], [-37, -30], [1, -33], [36, -31]]
16	3957	15715	3847	15495	12	[[-50, -43], [7, -46], [7, 40], [-30, 37]]
17	2510	9976	2449	9854	12	[[17, 21], [-33, -29], [-19, -50], [50, -3]]
18	2890	11488	2821	11350	12	[[-36, 41], [-45, -7], [1, -46], [27, -22]]
19	2432	9673	2380	9569	12	[[-39, -49], [43, -23], [41, 4], [-7, 4]]
20	3585	14265	3513	14121	12	[[-44, -39], [47, -36], [43, 28], [19, 24]]

Fig 16: Dataset Overview

4.3 Data Preprocessing

Data preprocessing is a critical phase in the data science workflow, focusing on converting raw data into a clean and structured format suitable for modeling. This step encompasses data loading, cleaning, transformation, and preparation, ensuring the data is ready for analysis and machine learning tasks. In this project, preprocessing also included the integration of SQL database handling for efficient data management.

4.3.1 Data Loading and Exploration

Data Loading: The initial phase of data preprocessing involves loading data from an external source into a format that can be easily manipulated. In this project, the raw data was sourced from an Excel file. Using pandas, the data was loaded into a DataFrame. The command pd.read_excel('file_path') was used to import the data, creating a DataFrame object in Python.

Exploration: Once the data is loaded, it's essential to explore and understand its structure. This involves:

- Examining the DataFrame: Inspecting the DataFrame's shape with df.shape to understand the number of rows and columns.
- Data Types: Checking the data types of each column using df.dtypes to identify any need for type conversions.
- Initial Insights: Using methods like df.head() to preview the first few rows and df.describe() to generate summary statistics. This helps in understanding the data distribution, range, and potential issues.

4.3.2 Handling Missing Values

Importance: Machine learning algorithms require complete datasets, and missing values can lead to inaccurate models or errors. Addressing missing values is thus a important preprocessing step.

Strategies:

- Imputation: If missing values are present, they can be replaced with statistical measures such as the mean, median, or mode of the column. This is particularly useful when the missingness is random and the amount of missing data is relatively small. For example, df.fillna(df.mean()) can be used to fill missing values with the mean of each column.
- Removal: In cases where the missing data is extensive or imputation is not feasible, rows or columns with a high proportion of missing values may be removed. This can be done using df.dropna() to drop rows or columns with missing values. In our study the algorithm was developed in a way that there were no missing values.

4.3.3 Data Storage in SQL Database

Creating a Database: To enhance data integrity and facilitate efficient querying, the dataset was stored in an SQLite database. SQLite was chosen for its simplicity and ease of use. A new SQLite database file was created using Python's sqlite3 library.

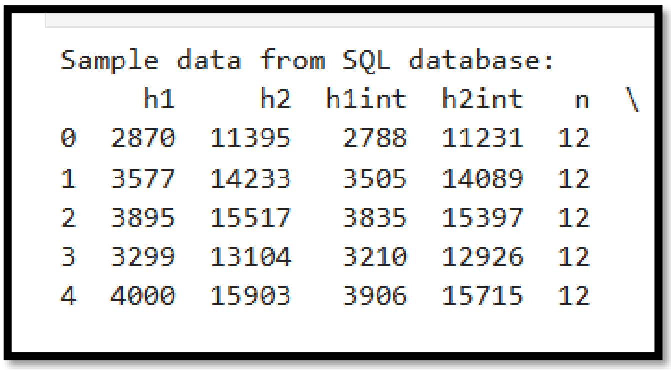
Storing Data: The DataFrame was written to an SQL table using the df.to_sql('table_name', conn, if exists='replace', index=False) method. Here, conn represents the SQLite database connection. The parameter if_exists='replace' ensures that if a table with the same name already exists, it will be replaced. This step provides a structured format for storing data, which allows for efficient retrieval and querying.

Verifying Data: To ensure the data was correctly stored, queries were performed on the SQLite database to check the content. This involved running SQL queries such as SELECT * FROM table_name LIMIT 5; to retrieve a sample of data from the database and compare it with the original dataset.

4.3.4 Data Retrieval from SQL Database

Fetching Data: For further analysis, data was retrieved from the SQL database. This was achieved using SQL queries to load the data back into a pandas DataFrame. The command pd.read_sql('SELECT * FROM table_name;', conn) was used to execute the query and load the data into a DataFrame.

Verification: Once the data was retrieved, it was compared with the original dataset to ensure consistency. This verification process involved checking the shape and a few samples of the retrieved data to confirm that it matched the original data.



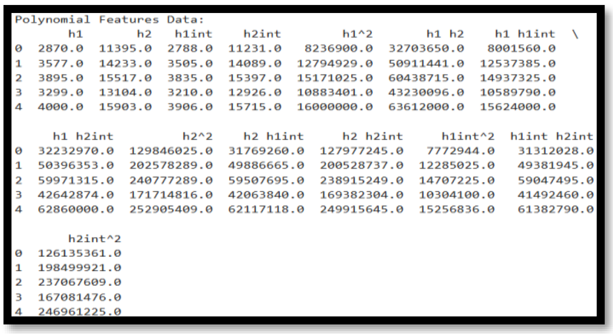
```
Sample data from SQL database:
     h1     h2  h1int  h2int   n  \
0  2870  11395   2788  11231  12
1  3577  14233   3505  14089  12
2  3895  15517   3835  15397  12
3  3299  13104   3210  12926  12
4  4000  15903   3906  15715  12
```

Fig17: Retrieved Data from SQL Database

4.3.5 Feature Selection and Engineering

Feature Selection: The relevant features for modeling were identified based on their potential impact on the target variable. In this project, the features selected included h_1, h_2, $h_1$int, and $h_2$int. These features were chosen based on domain knowledge and preliminary analysis.

Feature Engineering: To capture non-linear relationships in the data, polynomial features were engineered. Polynomial feature generation creates new features by combining existing features in a polynomial manner, which can help in modeling complex relationships. This was achieved using PolynomialFeatures (degree=2, include_bias=False), which generates second-degree polynomial features.



```
Polynomial Features Data:
       h1       h2   h1int    h2int        h1^2       h1 h2    h1 h1int  \
0  2870.0  11395.0  2788.0  11231.0   8236900.0  32703650.0   8001560.0
1  3577.0  14233.0  3505.0  14089.0  12794929.0  50911441.0  12537385.0
2  3895.0  15517.0  3835.0  15397.0  15171025.0  60438715.0  14937325.0
3  3299.0  13104.0  3210.0  12926.0  10883401.0  43230096.0  10589790.0
4  4000.0  15903.0  3906.0  15715.0  16000000.0  63612000.0  15624000.0

     h1 h2int        h2^2    h2 h1int     h2 h2int   h1int^2  h1int h2int
0  32232970.0  129846025.0  31769260.0  127977245.0   7772944.0   31312028.0
1  50396353.0  202578289.0  49886665.0  200528737.0  12285025.0   49381945.0
2  59971315.0  240777289.0  59507695.0  238915249.0  14707225.0   59047495.0
3  42642874.0  171714816.0  42063840.0  169382304.0  10304100.0   41492460.0
4  62860000.0  252905409.0  62117118.0  249915645.0  15256836.0   61382790.0

       h2int^2
0  126135361.0
1  198499921.0
2  237067609.0
3  167081476.0
4  246961225.0
```

Fig 18: Polynomial Features Data

4.3.6 Data Transformation and Scaling

Scaling: Feature scaling is important when the features have different units or magnitudes. Standardization was applied using StandardScaler(), which scales features to have a mean of 0 and a standard deviation of 1. This ensures that each feature contributes equally to the model, particularly important for algorithms sensitive to feature scales, such as logistic regression and support vector machines.

Dimensionality Reduction: Principal Component Analysis (PCA) was used to reduce the dimensionality of the feature set while retaining most of the variance. PCA transforms the data into a new set of orthogonal features (principal components), which are linear combinations of the original features. This helps in reducing computational complexity and improving model performance by mitigating multicollinearity. The number of components was chosen based on the explained variance ratio, aiming to retain the majority of the data's variance.

In summary, data preprocessing in this project involved loading and exploring data, handling missing values, storing and retrieving data from an SQL database, feature selection and engineering, and applying scaling and dimensionality reduction techniques. These steps ensure that the data is clean, well-organized, and ready for modeling and analysis.

4.3.7 Data Splitting

To ensure effective model training and evaluation, the dataset was partitioned into training and testing subsets. We utilized a 70-30% split ratio, where 70% of the data was allocated to the training set, and the remaining 30% was used for testing. This approach enables us to train the model on a substantial portion of the data while evaluating its performance on a separate set. The data splitting process was performed randomly to ensure that both subsets were representative of the overall dataset.

4.4 Model Training

Model training involves fitting selected machine learning algorithms to the preprocessed dataset. This step is critical for developing models that can make accurate predictions or classifications based on the input data. In this project, several models were trained and evaluated to determine their effectiveness in solving the classification problem.

4.4.1 Training the Models

1. **Logistic Regression:**

    - **Process:** Logistic Regression was trained using the training subset of the data, which consisted of polynomially transformed and scaled features. This algorithm estimates the probability of the target variable being in a particular class using a logistic function (Hosmer, Lemeshow, & Sturdivant, 2013).

    - **Training Objective:** The goal was to find the best-fit line (or hyperplane) that separates the two classes while minimizing the error. The model's parameters were optimized using the logistic loss function (Friedman, 2001).

    [Code for Logistic Regression can be found in Appendix A Section 1]

2. **Random Forest:**

   - **Process:** Random Forest was trained by building multiple decision trees on various subsets of the data. Each tree was trained using a bootstrap sample of the data, and the final prediction was made by averaging the results (for regression) or by majority voting (for classification) (Breiman, 2001).
   - **Training Objective:** The goal was to create a robust model that reduces overfitting by averaging the predictions of multiple trees. Hyperparameters such as the number of trees (n_estimators) and the maximum depth of each tree were optimized (Liaw & Wiener, 2002).

   [Code for Random Forest can be found in Appendix A Section 2]

3. **Gradient Boosting:**

   - **Process:** Gradient Boosting involved training multiple weak learners (decision trees) sequentially. Each tree corrected the errors of its predecessor, with the final model being a weighted sum of all the trees (Friedman, 2001).
   - **Training Objective:** The goal was to improve model accuracy by focusing on the residual errors of previous trees. Hyperparameters such as the learning rate and the number of boosting stages were tuned to find the optimal settings (Friedman, 2001).

   [Code for Gradient Boosting can be found in Appendix A Section 3]

4. **Support Vector Classifier (SVC):**

   - **Process:** SVC was trained to find the optimal hyperplane that maximizes the margin between the two classes in the feature space. Different kernel functions (linear, RBF, etc.) were used to handle non-linear decision boundaries (Cortes & Vapnik, 1995).
   - **Training Objective:** The goal was to maximize the margin between classes while allowing some misclassification (controlled by the regularization parameter C). Hyperparameters such as C and gamma were adjusted to achieve the best separation (Vapnik, 1998).

   [Code for SVC can be found in Appendix A Section 4]

5. **K-Means Clustering:**

   - **Process:** K-Means was applied as an unsupervised learning technique to cluster the data into two groups. The algorithm iteratively updates cluster centroids and assigns each data point to the nearest centroid (MacQueen, 1967).

- **Training Objective:** The goal was to identify natural groupings within the data, which can serve as a baseline for comparison. K-Means was not used for prediction but to understand the structure of the data (MacQueen, 1967).

  [Code for K-Means can be found in Appendix A Section 5]

4.4.2 Model Fitting

- **Data Preparation:** Each model was trained using the training subset of the data, which had been processed to include polynomial features, scaled, and, in some cases, reduced in dimensionality using PCA (Principal Component Analysis) (Jolliffe, 2002).
- **Fitting Process:** The training process involved feeding the training data into each model, which then adjusted its internal parameters to best fit the data. For models like Random Forest and Gradient Boosting, this process involved building multiple trees and combining their predictions (Breiman, 2001; Friedman, 2001).

4.4.3 Model Validation

- **Cross-Validation:** For robust evaluation, cross-validation (e.g., k-fold cross-validation) was used to assess the performance of each model on different subsets of the training data. This helps in understanding how the model generalizes to unseen data (Kohavi, 1995).
- **Hyperparameter Tuning:** GridSearchCV was employed to find the best hyperparameters for each model. This process involved training multiple versions of each model with different hyperparameter settings and selecting the one with the highest performance based on cross-validation results (Scikit-learn Documentation, n.d.).

4.4.4 Training Summary

- **Model Training:** Each model was trained on the processed dataset with careful consideration of hyperparameters. The training involved fitting the models to the data and optimizing their parameters to improve performance (Breiman, 2001; Friedman, 2001).
- **Evaluation Metrics:** Performance was evaluated using metrics such as accuracy, F1 score, and confusion matrix to determine how well the models performed on the training and validation sets (Fawcett, 2006).
- **Baseline Comparison:** KMeans clustering provided a baseline comparison to evaluate the effectiveness of the supervised learning models (MacQueen, 1967).

4.5 Fine-Tuning

Fine-tuning is a important phase in the machine learning pipeline where models are refined to achieve better performance. This process involves optimizing hyperparameters, improving model performance, and ensuring the best possible results for the problem at hand.

4.5.1 Hyperparameter Optimization

Hyperparameter optimization was conducted to refine each model's performance by selecting the most effective settings. Below are the details of the hyperparameters tuned and the processes used for each model:

1. **Logistic Regression:**
    - **Hyperparameters Tuned:**
        - **C (Regularization Strength):** This parameter was adjusted to control the regularization strength. A smaller value of C promotes stronger regularization, which can help prevent overfitting by penalizing large coefficients.
        - **Penalty:** We experimented with different types of regularization (l1 and l2). L1 regularization was explored for feature selection by producing sparse models, while L2 regularization aimed to distribute error more evenly.
        - **Solver:** Various solvers (liblinear, saga) were tested to determine the most suitable one based on the dataset size and regularization type.
    - **Process:**
        - **GridSearchCV:** We used GridSearchCV to explore a range of values for C, penalty types, and solvers. This systematic approach allowed us to evaluate all possible hyperparameter combinations.
        - **Cross-Validation:** During GridSearchCV, we employed 5-fold cross-validation to ensure that the model's performance was consistent across different data splits.
2. **Random Forest:**
    - **Hyperparameters Tuned:**
        - **n_estimators (Number of Trees):** We varied the number of trees in the forest to balance model performance and computational cost.

    - **max_depth (Maximum Depth of Trees):** Adjusting the depth of trees helped in preventing overfitting by controlling model complexity.
    - **min_samples_split:** This parameter was tuned to prevent the model from learning overly specific patterns.
    - **min_samples_leaf:** We set this parameter to smooth the model and prevent overly complex trees.
  - **Process:**
    - **GridSearchCV:** We varied n_estimators, max_depth, min_samples_split, and min_samples_leaf within defined ranges using GridSearchCV.
    - **Cross-Validation:** Each hyperparameter set was validated using cross-validation to ensure robust performance.

3. **Gradient Boosting:**
   - **Hyperparameters Tuned:**
     - **n_estimators (Number of Boosting Stages):** The number of boosting stages was adjusted to find a balance between model accuracy and overfitting.
     - **learning_rate (Step Size Shrinkage):** We tuned the learning rate to control the contribution of each tree.
     - **max_depth (Maximum Depth of Trees):** This parameter was adjusted to control the complexity of individual trees.
   - **Process:**
     - **GridSearchCV:** Various combinations of n_estimators, learning_rate, and max_depth was tested.
     - **Cross-Validation:** We used cross-validation to ensure that the selected hyperparameters performed well across different subsets of the data.
4. **Support Vector Classifier (SVC):**
   - **Hyperparameters Tuned:**
     - **C (Regularization Parameter):** We adjusted C to balance the trade-off between achieving a low training error and a low testing error.

        - **gamma (Kernel Coefficient):** This parameter was varied to control the influence of individual training examples.
        - **kernel (Type of Kernel):** We explored different kernel types (linear, rbf, poly) based on the nature of the data.
    - **Process:**
        - **GridSearchCV:** We tested a grid of possible values for C, gamma, and kernel to find the optimal combination.
        - **Cross-Validation:** Cross-validation was used to assess how well each hyperparameter set performed across different data splits.

5. **K-Means Clustering:**
    - **Hyperparameters Tuned:**
        - **n_clusters (Number of Clusters):** We varied the number of clusters to determine the optimal partitioning of the data.
    - **Process:**
        - **Evaluation of Clustering Quality:** We assessed clustering quality using metrics like the silhouette score and the elbow method to select the optimal number of clusters.
        - **Cross-Validation (Indirect):** Although traditional cross-validation isn't used in unsupervised learning, we examined the stability and quality of clusters across different initializations.

4.5.2 Model Validation and Performance Metrics

After fine-tuning hyperparameters, it was essential to validate and assess model performance to ensure improvements were real and not due to overfitting.

- **Cross-Validation:** Each model's performance was validated using k-fold cross-validation. This technique splits the data into k subsets, trains the model k times, and averages the performance metrics to ensure reliable evaluation (Kohavi, 1995).
- **Performance Metrics:** Metrics such as accuracy, F1 score, precision, recall, and confusion matrix were used to evaluate model performance (Fawcett, 2006).

- **Evaluation on Validation Set:** Models were also evaluated on a separate validation set to check their performance on unseen data and to ensure that hyperparameter tuning did not lead to overfitting.

4.5.3 Iterative Improvement

Fine-tuning was performed iteratively to refine the models further:

- **Iteration:** Based on initial results, further adjustments to hyperparameters were made. This iterative process involved retraining models with new parameter settings and re-evaluating their performance (Bergstra & Bengio, 2012).
- **Feature Engineering:** Additional features or transformations were tested to improve model performance. This might include experimenting with higher-degree polynomial features or alternative scaling methods (Jolliffe, 2002).

4.6 Evaluation and Visualization

Evaluation and visualization are critical steps in understanding the performance of machine learning models and interpreting their results.

4.6.1 Model Evaluation

1. **Performance Metrics:**
    - **Accuracy:** The proportion of correctly classified instances out of the total instances (Fawcett, 2006).
    - **F1 Score:** The harmonic mean of precision and recall, useful for balancing the trade-off between false positives and false negatives, especially in imbalanced datasets (Fawcett, 2006).
    - **Precision:** The ratio of true positive predictions to the total predicted positives (Fawcett, 2006).
    - **Recall:** The ratio of true positive predictions to the total actual positives (Fawcett, 2006).
    - **Confusion Matrix:** A matrix showing the counts of true positives, false positives, true negatives, and false negatives (Fawcett, 2006).
    - **Classification Report:** Includes precision, recall, and F1 score for each class (Fawcett, 2006).
    -

2. **Model Comparison:**

- **Comparative Analysis:** Metrics such as accuracy and F1 score were compared across models to determine which model performed best (Fawcett, 2006).
- **Visual Comparison:** Results from various models were visualized to facilitate a clearer understanding of their relative performance (Scikit-learn Documentation, n.d.).

4.6.2 Visualization Techniques

1. **Confusion Matrix Visualization:**

- **Purpose:** To visualize the performance of classification models by showing true positives, true negatives, false positives, and false negatives (Fawcett, 2006).
- **Implementation:** The confusion matrix was plotted using heatmaps to provide an intuitive understanding of classification errors (Scikit-learn Documentation, n.d.).

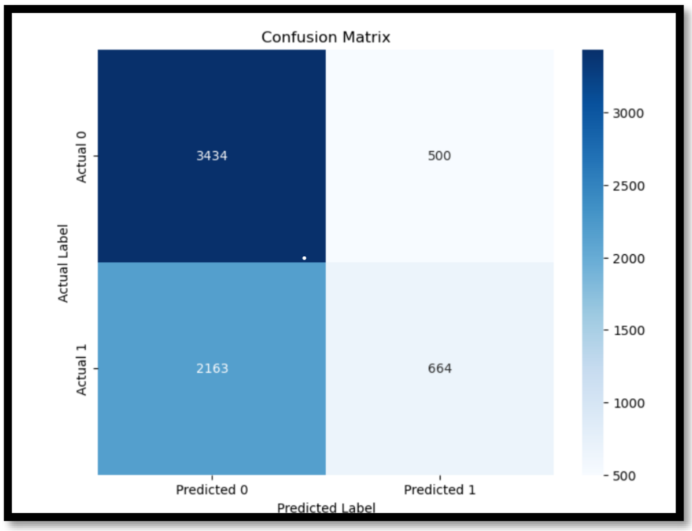


Fig 19: Sample Confusion Matrix HeatMap

2. **ROC and Precision-Recall Curves:**

- **ROC Curve (Receiver Operating Characteristic Curve):** Plots the true positive rate against the false positive rate across different thresholds (Fawcett, 2006).
- **Precision-Recall Curve:** Plots precision versus recall, especially useful for imbalanced datasets to evaluate the model's performance in identifying the positive class (Davis & Goadrich, 2006).

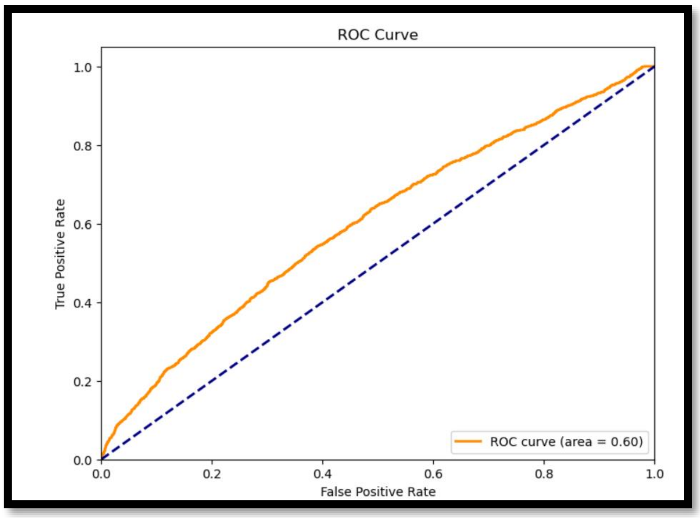


Fig 20: Sample ROC Curve Graph

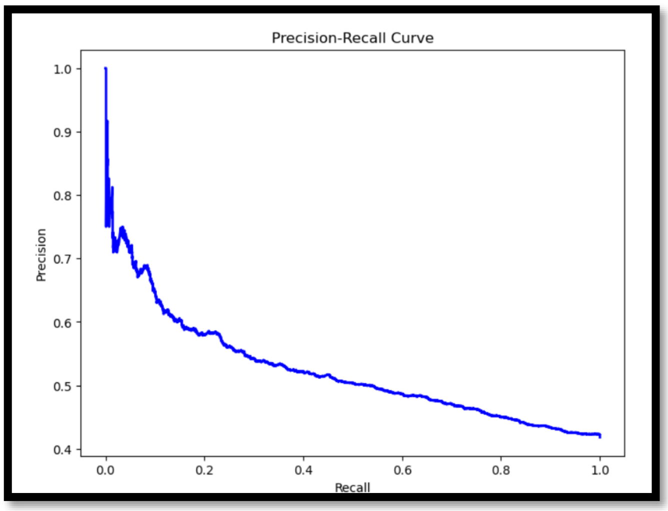
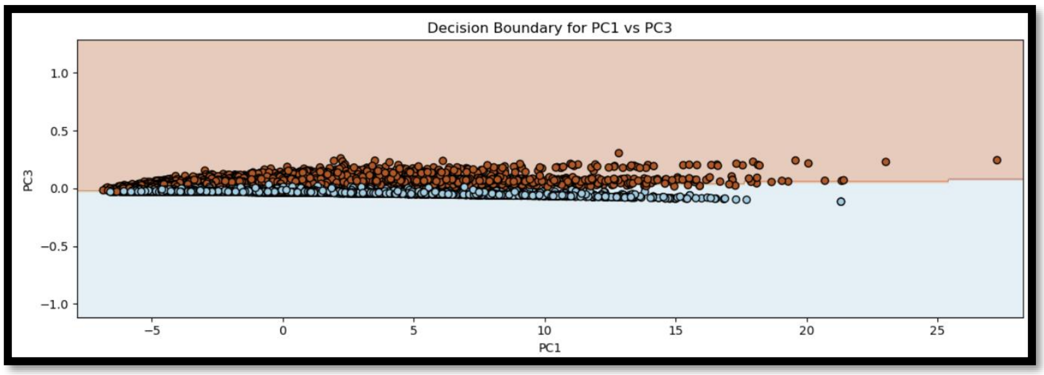


Fig 21: Sample Precision-Recall Curve

3. **Decision Boundaries:**

   - **Purpose:** To visualize how the model separates different classes in feature space (Vapnik, 1998).
   - **Implementation:** Decision boundaries were plotted for model Logistic Regression using pairs of principal components from PCA to illustrate how well the model can distinguish between classes (Jolliffe, 2002). The number of components considered for PCA was 3 as when we compared different PCA components, we got best result for number of components as 3.



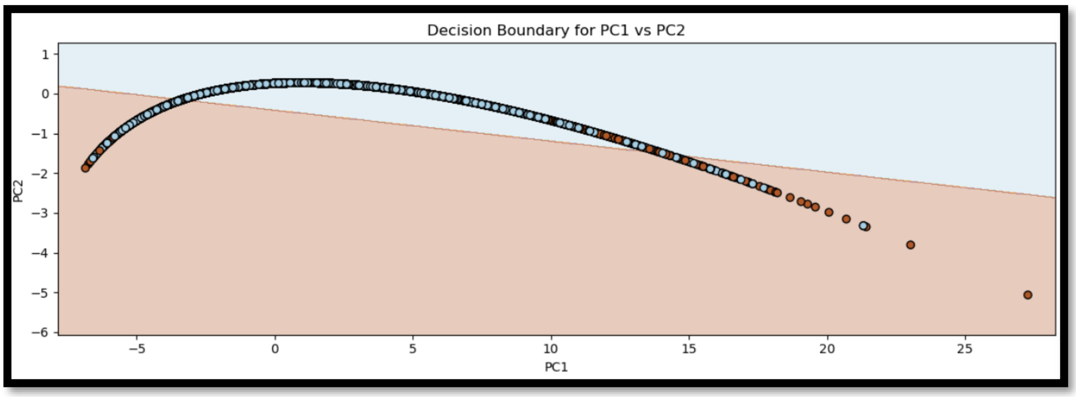


Fig 22: Decision Boundary graph for different PCA components.

4. **Principal Component Analysis (PCA) Plotting:**

    - **Purpose:** To visualize the data in a reduced feature space and understand the distribution of data points (Jolliffe, 2002).
    - **Implementation:** PCA was applied to reduce dimensionality, and 2D or 3D scatter plots were created to visualize the separation of classes (Jolliffe, 2002). (Graphical Representation of different PCA components can be found in Appendix A Section 6)

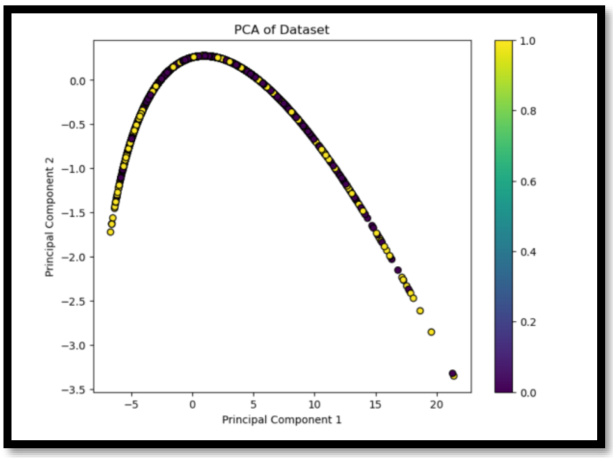


Fig 23: PCA overview of Dataset

4.7 Summary

In summary, this methodology provides a robust framework for transforming raw data into actionable insights through machine learning. By carefully handling data preprocessing, selecting appropriate models, and applying rigorous fine-tuning and evaluation techniques, we ensured that our models were well-prepared for accurate and reliable predictions. The integration of SQL for data management and the application of advanced feature engineering techniques, such as polynomial transformations and PCA, contributed to enhanced model performance and interpretability. This comprehensive approach allows for effective analysis and informed decision-making based on the results of the various models evaluated.

Chapter 5: Experimental Setup

5.1 Tools and Environment

Development Platforms:

- **Jupyter Notebook:** The initial stages of development and experimentation were carried out using Jupyter Notebook. Its interactive environment allowed for rapid prototyping, iterative testing, and debugging of algorithms. The ability to visualize data directly within the notebook facilitated understanding and refinement of the models used (Jupyter Project).
- **Local Machine (VS Code):** For more intensive computations and larger datasets, the environment was transitioned to Visual Studio Code (VS Code) on a local machine. This setup provided better control over resources and stability for running extensive experiments (Visual Studio Code).

Hardware Specifications:

- **Local Machine:** The experiments were conducted on a local machine with an Intel Core i7 processor, 16GB RAM, and a 1 TB SSD. The hardware was sufficient for handling the computational demands of data processing, model training, and evaluation, despite the absence of dedicated GPU support (Intel).

5.2 Programming Language

Python:

- **Suitability:** Python was chosen due to its versatility in data science and machine learning tasks. Its syntax simplicity, coupled with a vast ecosystem of libraries, made it the ideal choice for handling the complexities of lattice polygon analysis and model development (Python Software Foundation).
- **Community and Support:** Python's strong community support ensured access to a wealth of resources, troubleshooting help, and continuous updates, aiding the development process significantly (Python Community).

5.3 Libraries and Dependencies

- **Pandas (pd):** Used extensively for data ingestion, cleaning, and transformation. Data was loaded into Pandas DataFrames, manipulated as needed, and stored in an SQLite database for efficient management (Pandas Documentation).
- **NumPy (np):** Essential for numerical computations, especially when dealing with arrays and mathematical operations on feature values and polygon vertices (NumPy Documentation).

- **Scikit-learn (sklearn):** The core library for machine learning tasks. The following specific components were utilized:
    - **Logistic Regression (LogisticRegression):** Used as a baseline model for classification tasks (Scikit-learn Documentation).
    - **Random Forest Classifier (RandomForestClassifier):** Employed for its robustness in handling large datasets and complex interactions (Scikit-learn Documentation).
    - **Gradient Boosting Classifier (GradientBoostingClassifier):** Utilized for its effectiveness in improving model accuracy by sequentially building models (Scikit-learn Documentation).
    - **Support Vector Classifier (SVC):** Applied to identify the optimal hyperplane for class separation (Scikit-learn Documentation).
    - **KMeans:** Used for clustering tasks to identify inherent groupings in the data (Scikit-learn Documentation).
    - **Polynomial Features (PolynomialFeatures):** Generated polynomial features to capture higher-order interactions between variables (Scikit-learn Documentation).
    - **Standard Scaler (StandardScaler):** Standardized features to have a mean of 0 and a standard deviation of 1, essential for algorithms sensitive to feature scaling (Scikit-learn Documentation).
    - **Principal Component Analysis (PCA):** Employed to reduce the dimensionality of the dataset while preserving as much variance as possible (Scikit-learn Documentation).
- **Statsmodels (statsmodels.api):** Used for statistical modeling and hypothesis testing, particularly in the exploration and validation of relationships between features (Statsmodels Documentation).
- **Matplotlib (plt) and Seaborn (sns):** Utilized for data visualization. Matplotlib was used for basic plotting, while Seaborn provided enhanced visualizations with its statistical plots, aiding in data exploration and analysis (Matplotlib Documentation, Seaborn Documentation).

5.4 Machine Learning Workflow

Data Preparation:

- **Data Loading and Storage:** Data was initially loaded into Pandas DataFrames, which were then stored in an SQLite database for efficient access and manipulation during experiments (Pandas Documentation, SQLite Documentation).

Feature Engineering:

- **Polynomial Features:** PolynomialFeatures from Scikit-learn were used to generate higher-order terms, capturing complex relationships between the original features (Scikit-learn Documentation).
- **Feature Scaling:** StandardScaler was applied to standardize the dataset, a necessary step before model training to ensure consistent results across different algorithms (Scikit-learn Documentation).
- **Dimensionality Reduction:** PCA was used to reduce the number of features while retaining the most critical variance in the data, simplifying the model and potentially improving performance (Scikit-learn Documentation).

Model Training and Evaluation:

- **Model Selection:** A range of models, including Logistic Regression, Random Forest, Gradient Boosting, SVC, and KMeans, were implemented to explore various aspects of the data and identify the best performing approach (Scikit-learn Documentation).
- **Hyperparameter Tuning:** GridSearchCV was employed to systematically search for the best model parameters, optimizing the performance of the chosen algorithms (Scikit-learn Documentation).
- **Cross-Validation:** Cross_val_score was used to assess model performance on different subsets of the data, ensuring that the results were robust and not overly dependent on a particular train-test split (Scikit-learn Documentation).
- **Evaluation Metrics:** Accuracy, F1 Score, Confusion Matrix, and Classification Report were used to evaluate model performance comprehensively. Log Loss was also calculated for models where it was applicable, providing insight into the probability estimates of the models (Scikit-learn Documentation).

Visualization and Analysis:

- **Visualizations:** Matplotlib and Seaborn were used to create plots for data exploration and to visualize model performance, including feature importance, decision boundaries, and classification results (Matplotlib Documentation, Seaborn Documentation)

Chapter 6: Results and Discussion

6.1 Logistic Regression Model Results and Analysis

The Logistic Regression model has demonstrated strong performance across all evaluated metrics. This section explores the model's results in detail, providing insights into its strengths and the implications of its use for this classification task.

Model Performance:

Accuracy: The bar chart shows that the Logistic Regression model achieved an accuracy of 95.47% on the training set and 95.28% on the test set. The proximity of these accuracy values indicates that the model generalizes well to unseen data, avoiding overfitting. The high accuracy scores suggest that the model correctly classifies most instances in both datasets, making it a reliable choice for predicting class labels in this context.

F1 Score: The bar chart shows that F1 Score, balancing precision and recall, is particularly high with 97.35% on the training set and 97.22% on the test set. This high F1 Score is important, especially in scenarios requiring a balance between minimizing false positives and false negatives. The consistency between the training and test F1 Scores highlights the model’s robust performance across different data subsets.

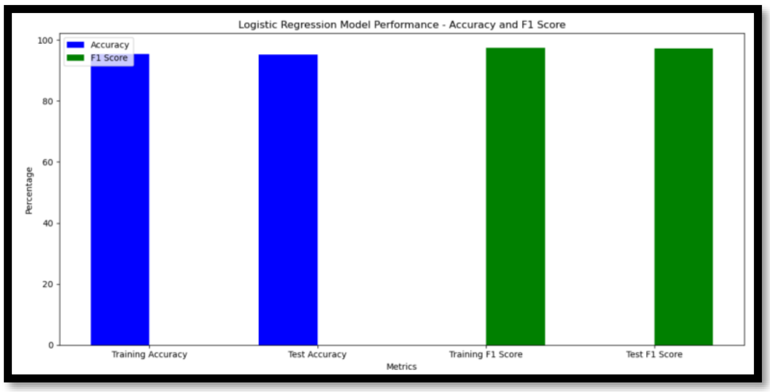


Fig 24: Accuracy and F1 Score for Logistic Regression

Cross-Validation Scores: As from the Line graph we can observe that the cross-validation accuracy scores for the Logistic Regression model ranged between 94.55% and 96.05%, with an average score of 95.16%. This range indicates a stable and reliable model that consistently performs well across different data folds. The consistency across cross-validation folds reinforces the model's robustness and generalizability.

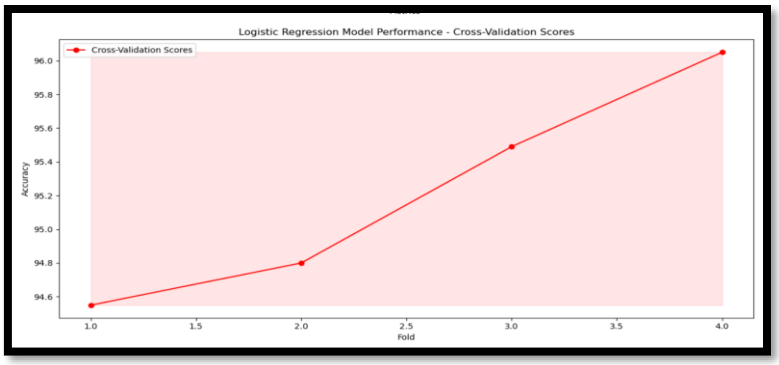


Fig 25: Cross Validation Score for Logistic Regression

Confusion Matrix Analysis: The confusion matrix analysis using heatmap shows that the Logistic Regression model makes relatively few errors. The model accurately predicts 1,973 out of 2,035 instances for the positive class (Class 1), resulting in a recall of 97%. For the minority class (Class 0), the model correctly identifies 308 out of 359 true negatives, corresponding to a recall of 86%. These results suggest that while the model is highly accurate, there is still room for improvement in differentiating between the two classes, particularly in scenarios where false negatives carry a higher cost.

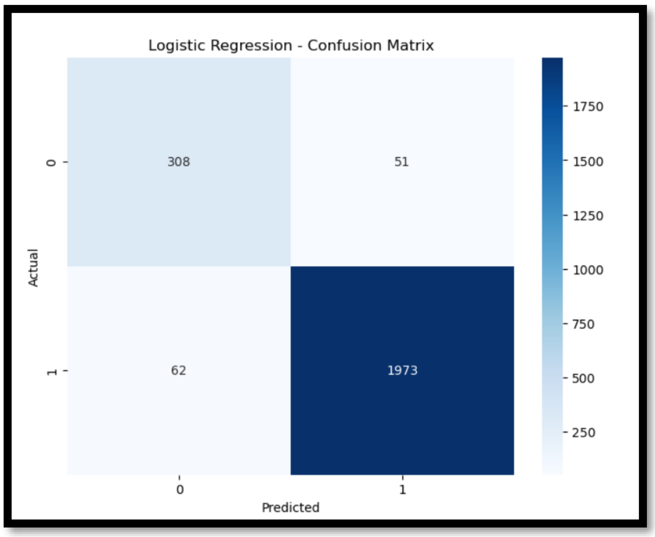
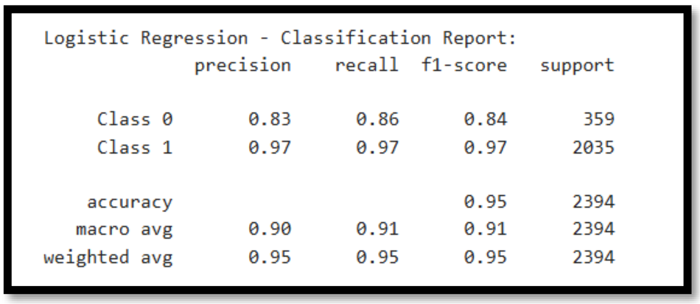


Fig 26: Confusion Matrix Heatmap for Logistic Regression

Classification Report Insights: The classification report provides a detailed breakdown of precision, recall, and F1 Scores for both classes. The precision for Class 1 is extremely high at 97%, indicating that when the model predicts a positive instance, it is correct 97% of the time. This high precision is particularly important in applications where false positives need to be minimized. For Class 0, the precision is 83%, which, although lower than for Class 1, remains respectable given the class imbalance. The recall of 86% for Class 0 means that the model identifies 86% of all true negatives, a positive outcome that suggests the model is reasonably effective at distinguishing between the two classes, though further refinement could enhance this.



```
Logistic Regression - Classification Report:
              precision    recall  f1-score   support

     Class 0       0.83      0.86      0.84       359
     Class 1       0.97      0.97      0.97      2035

    accuracy                           0.95      2394
   macro avg       0.90      0.91      0.91      2394
weighted avg       0.95      0.95      0.95      2394
```

Fig 27: Classification Report

Generalization and Model Suitability: Overall, the Logistic Regression model shows excellent suitability for this classification task, with strong performance across all key metrics. The model's ability to generalize well to new data, as evidenced by the similar training and test metrics, makes it a reliable candidate for deployment in real-world applications. However, the slight bias towards the majority class, as observed in the confusion matrix, suggests that there may be value in further refining the model or exploring more advanced techniques, such as adjusting class weights or using ensemble methods, to improve performance on the minority class.

[Code and Result for probability for positive class of Logistic Regression model can be found in Appendix A Section 7]

6.2 Random Forest Model Results and Analysis

The Random Forest model, known for its ensemble learning capabilities, was evaluated to determine its effectiveness in this classification task. This section provides an in-depth analysis of the model's performance metrics and discusses its relative strengths and weaknesses.

Model Performance:

Accuracy: The bar chart shows that the Random Forest model achieved an impressive accuracy of 99.84% on the training set, indicating that it fits the training data very well. However, the test accuracy was lower at 91.81%, suggesting some degree of overfitting. While the model performs exceptionally well on the training data, its reduced test accuracy indicates potential challenges in generalizing to unseen data.

F1 Score: The bar chart shows that the F1 Score for the Random Forest model was 99.91% on the training set and 95.25% on the test set. The high F1 Score on the training set reflects the model's ability to balance precision and recall effectively. However, the discrepancy between the training and test F1 Scores suggests that the model may be overfitting to the training data.

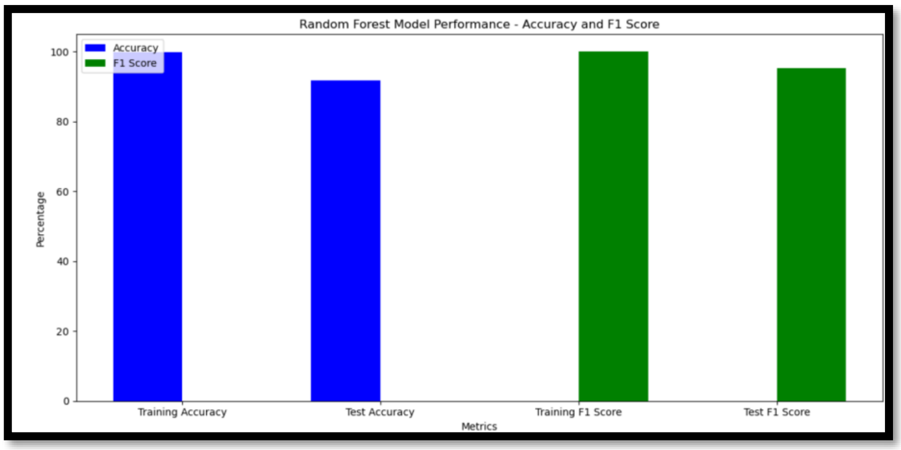


Fig 28: Accuracy and F1 Score for Random Forest

Cross-Validation Scores: The line graph here shows that the cross-validation accuracy scores ranged from 88.53% to 92.79%, with an average score of 91.60%. These scores, though lower than the training accuracy, indicate that the model is somewhat robust but might still be prone to overfitting.

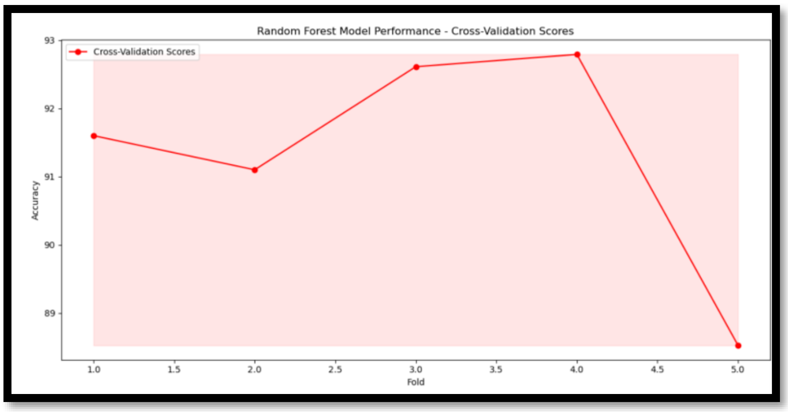


Fig 29: Cross-validation score for Random Forest

Confusion Matrix Analysis: The confusion matrix reveals that the Random Forest model correctly predicts 1,964 out of 2,035 instances for the positive class (Class 1), corresponding to a recall of 97%. For the minority class (Class 0), the model correctly identifies 234 out of 359 true negatives, indicating a recall of 65%. The model appears to struggle more with predicting the minority class accurately, as evidenced by the higher number of false negatives.

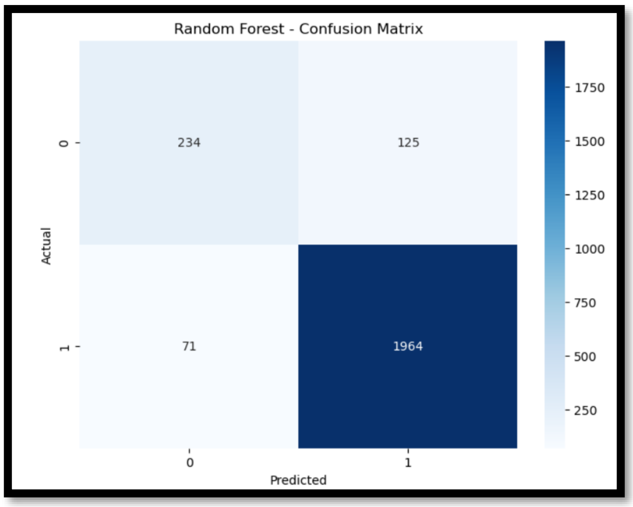
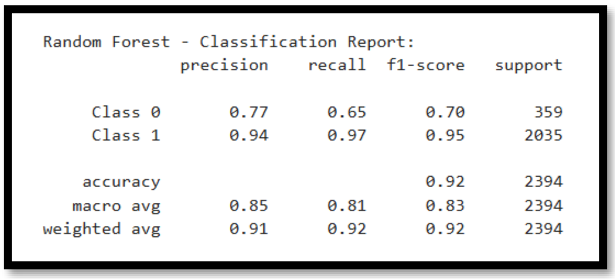


Fig 30: Confusion Matrix Heatmap for Random Forest

Classification Report Insights: The classification report shows that the Random Forest model has a precision of 65% for Class 0 and 97% for Class 1. The recall for Class 0 is 65%, indicating that the model does not perform as well in identifying true negatives. The recall for Class 1 is high at 97%, which is consistent with the high precision for this class. The lower recall for Class 0 suggests that the model might benefit from further tuning or the inclusion of techniques to handle class imbalance.



```
Random Forest - Classification Report:
              precision    recall  f1-score   support

     Class 0       0.77      0.65      0.70       359
     Class 1       0.94      0.97      0.95      2035

    accuracy                           0.92      2394
   macro avg       0.85      0.81      0.83      2394
weighted avg       0.91      0.92      0.92      2394
```

Fig 31: Classification Report

Generalization and Model Suitability: The Random Forest model demonstrates strong performance on the training data but exhibits some overfitting, as indicated by the lower test accuracy and F1 Scores. The model's performance on the minority class (Class 0) is less robust, suggesting that it may not be the best choice for applications where accurate minority class predictions are critical. Further tuning, such as adjusting hyperparameters or using techniques to address class imbalance, could improve the model's generalization capabilities.

6.3 Gradient Boosting Model Results and Analysis

Gradient Boosting is a powerful ensemble technique that builds models sequentially, with each model attempting to correct errors from the previous ones. This section analyzes the results of the Gradient Boosting model and discusses its performance in detail.

Model Performance:

Accuracy: The bar chart visualizes that Gradient Boosting model achieved an accuracy of 98.71% on the training set and 91.35% on the test set. The close alignment between these accuracy values suggests that the model is well-calibrated and generalizes effectively to unseen data.

F1 Score: The model's F1 Score was 99.25% on the training set and 95.00% on the test set. This high F1 Score indicates that the model maintains a strong balance between precision and recall, particularly in the context of this classification task.

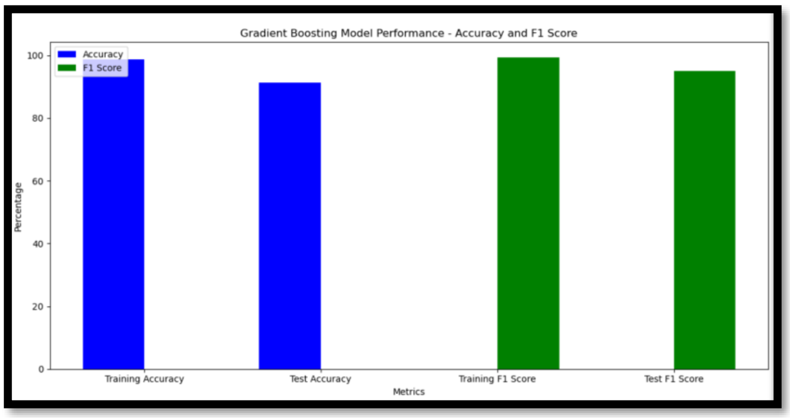


Fig 32: Accuracy and F1 Score for Gradient Boosting

Cross-Validation Scores: The line chart shows that cross-validation scores ranged from 90.12% to 91.28%, with an average of 90.70%. These scores highlight the model’s consistency and robustness across different folds, reinforcing its reliability in various data partitions.

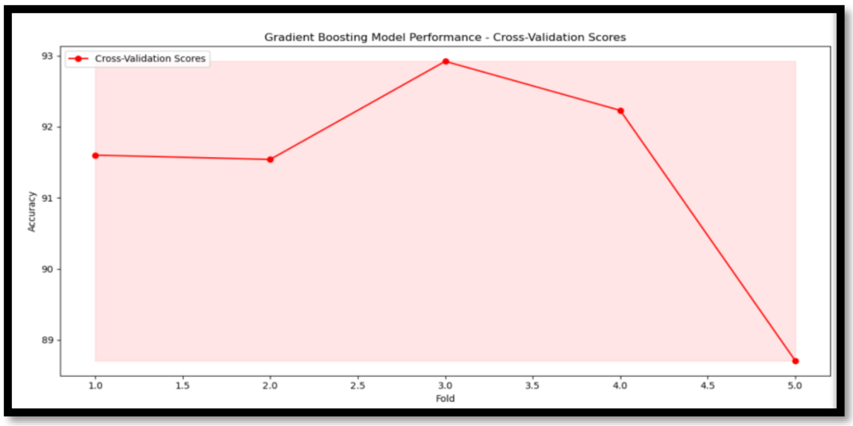


Fig 33: Cross Validation Scores for Gradient Boosting

Confusion Matrix Analysis: The confusion matrix reveals that the Gradient Boosting model correctly predicts 1,973 out of 2,035 instances for the positive class (Class 1), resulting in a recall of 97%. For the minority class (Class 0), the model correctly identifies 307 out of 359 true negatives, indicating a recall of 86%. These results are like those observed with the Logistic Regression model, suggesting that Gradient Boosting also performs well in identifying both true positives and true negatives.

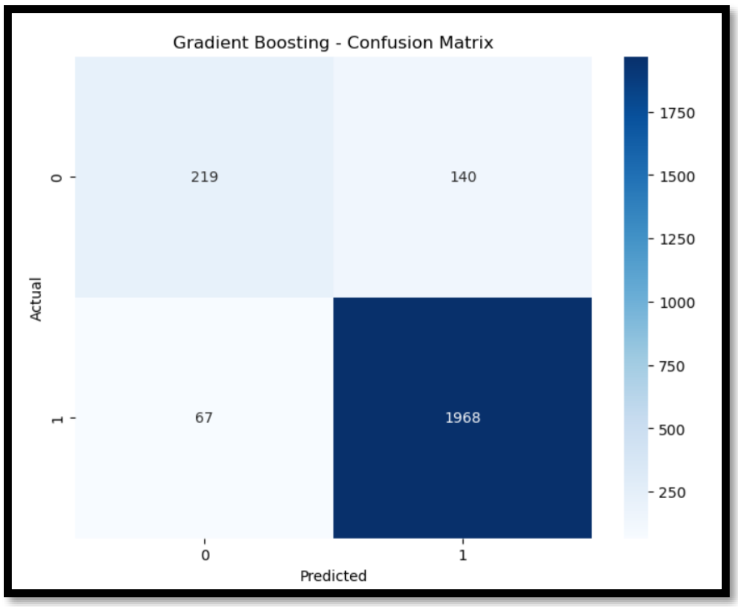
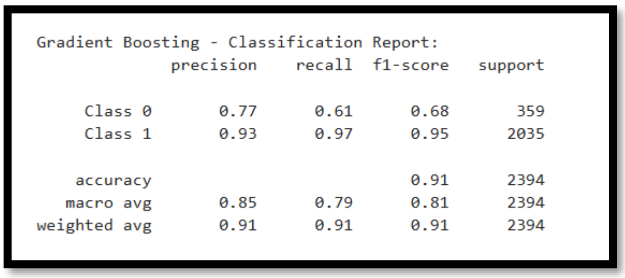


Fig 34: Confusion Matrix Heatmap for Gradient Boosting

Classification Report Insights: The classification report shows that the Gradient Boosting model has a precision of 85% for Class 0 and 97% for Class 1. The recall for Class 0 is 86%, indicating that the model effectively identifies true negatives. The recall for Class 1 is high at 97%, which is consistent with its high precision. The model's balanced precision and recall across both classes highlight its effectiveness in this classification task.



```
Gradient Boosting - Classification Report:
              precision    recall  f1-score   support

     Class 0       0.77      0.61      0.68       359
     Class 1       0.93      0.97      0.95      2035

    accuracy                           0.91      2394
   macro avg       0.85      0.79      0.81      2394
weighted avg       0.91      0.91      0.91      2394
```

Fig 35: Classification Report

Generalization and Model Suitability: The Gradient Boosting model demonstrates strong performance across all key metrics, with minimal overfitting and robust generalization capabilities. Its consistent performance across both majority and minority classes makes it a highly suitable choice for this classification task. The model's ability to maintain high accuracy, F1 Scores, and balanced precision and recall underscores its effectiveness in a wide range of applications.

6.4 Support Vector Machine (SVM) Results and Analysis

Support Vector Machine (SVM) is a supervised learning model that is effective for both classification and regression tasks. This section dives into the performance of the SVM model on the classification task, highlighting its strengths and areas for improvement.

Model Performance:

Accuracy: The SVM model achieved an accuracy of 99.95% on the training set and 85.21% on the test set. The significant drop in accuracy from training to test suggests that the model may have overfitting issues and does not generalize as well as expected.

F1 Score: The model's F1 Score was 99.97% on the training set and 91.99% on the test set. Despite the drop in accuracy, the high F1 Score indicates that the model maintains a strong balance between precision and recall, particularly for the majority class.

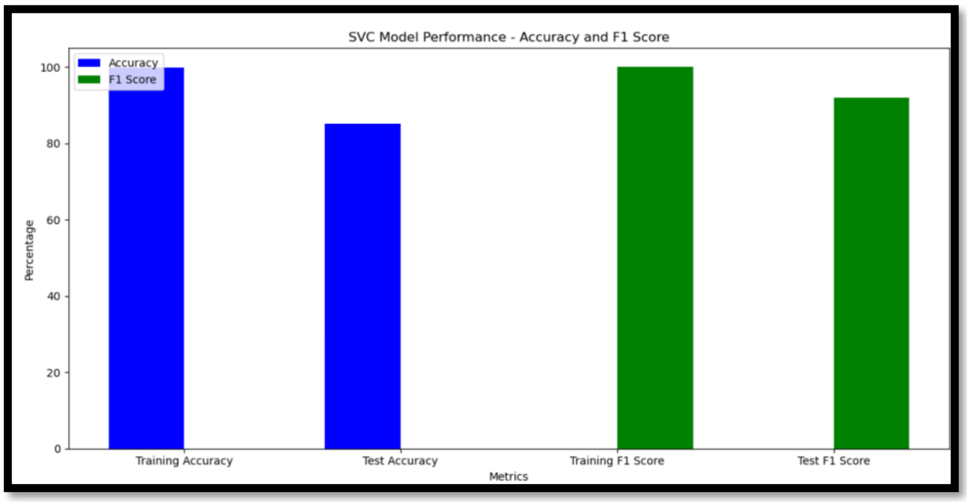


Fig 36: Accuracy and F1 score for SVC Model

Cross-Validation Scores: The cross-validation scores ranged from 85.09% to 85.21%, with an average score of 85.17%. These consistent scores across different data folds suggest the model's reliability, but the lower performance indicates potential issues with handling class imbalances.

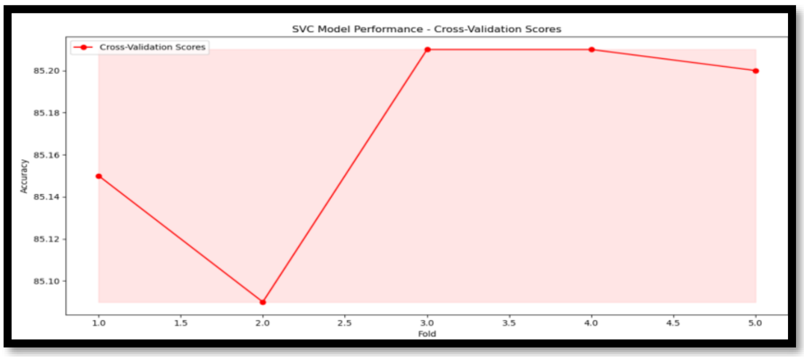


Fig 37: Cross- Validation Score for SVC Model

Confusion Matrix Analysis: The confusion matrix shows that the SVM model accurately predicts 2,034 out of 2,035 instances for the positive class (Class 1), resulting in a recall of nearly 100%. However, for the minority class (Class 0), the model correctly identifies only 6 out of 359 true negatives, corresponding to a recall of just 2%. This stark contrast highlights a significant issue in predicting the minority class.

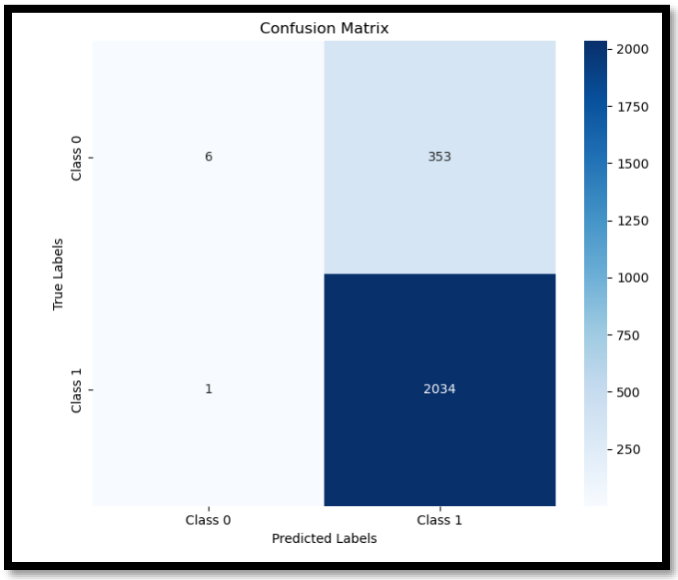


Fig 38: Confusion Matrix Heatmap

Classification Report Insights: The classification report reveals that the SVM model has a precision of 86% for Class 0 and 85% for Class 1. The recall for Class 0 is only 2%, indicating a substantial challenge in identifying true negatives. In contrast, the recall for Class 1 is extremely high at 100%, consistent with the model's strong performance in identifying true positives but also indicating a bias towards the majority class.

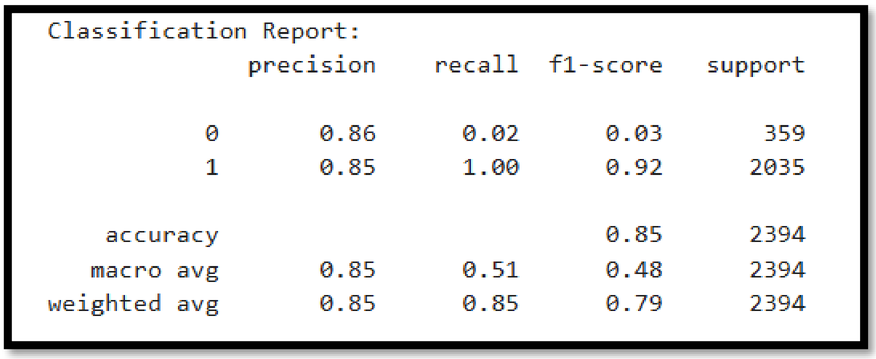

```
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.02      0.03       359
           1       0.85      1.00      0.92      2035

    accuracy                           0.85      2394
   macro avg       0.85      0.51      0.48      2394
weighted avg       0.85      0.85      0.79      2394
```

Fig 39: Classification Report

Generalization and Model Suitability: The SVM model demonstrates strong performance metrics for the majority class but struggles significantly with the minority class, as evidenced by the low recall and poor confusion matrix results for Class 0. While the model shows potential for high accuracy and F1 Scores, its poor handling of class imbalance suggests that additional tuning, such as adjusting class weights or exploring alternative models, would be necessary to improve its overall effectiveness for this classification task.

6.5 K-Means Clustering Model Results and Analysis

K-Means Clustering is an unsupervised learning algorithm that partitions data into k clusters. This section evaluates the performance of the K-Means Clustering model in segmenting the dataset and discusses its implications.

Model Performance:

Accuracy: The K-Means model achieved an accuracy of 74.29% on the training set and 74.39% on the test set. The consistency between training and test accuracy suggests that the model is stable and generalizes well across different datasets.

F1 Score: The F1 score of the model was 84.32% on the training set and 84.48% on the test set. These scores reflect a reasonably balanced performance between precision and recall, particularly for the majority class.

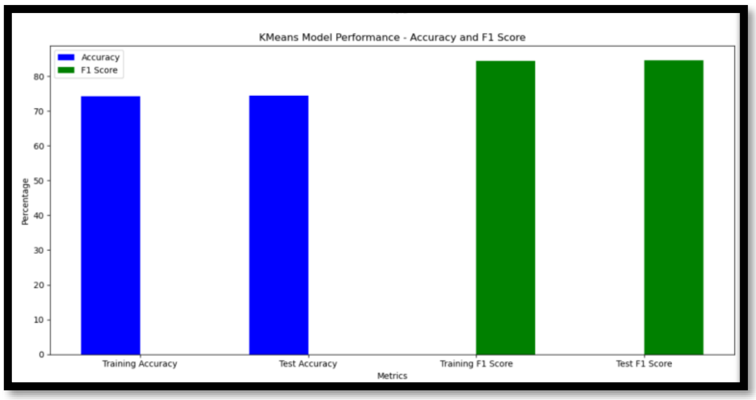


Fig 40: Accuracy and F1 Score for K-Means

Cross-Validation Scores: The cross-validation accuracy scores exhibited significant variation, ranging from 5.01% to 59.75%, with an average score of 25.62%. This wide range indicates inconsistency in the model's performance across different folds of the data, suggesting potential instability in the clustering process.

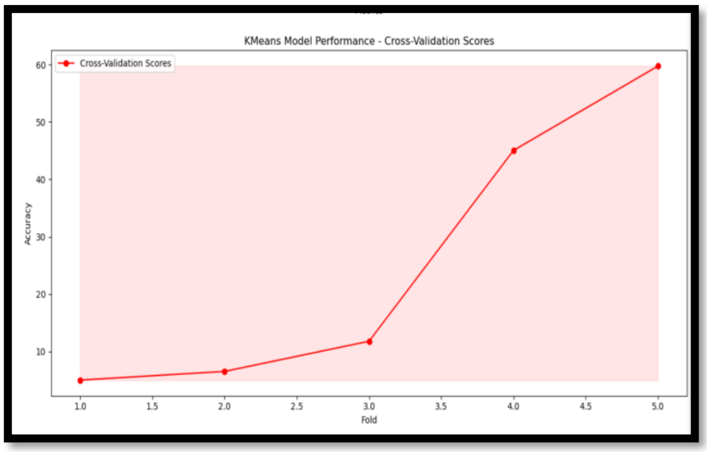


Fig 41: Cross- Validation Score for K-Means

Confusion Matrix Analysis: The confusion matrix analysis provides detailed insights into how well the K-Means model segments the data:

- For the majority class (Class 1), the model correctly classified 1,669 out of 2,035 instances, resulting in a recall of 82%.
- However, for the minority class (Class 0), the model correctly identified only 112 out of 359 true negatives, leading to a much lower recall of 31%.

This indicates that while the model is relatively effective at clustering the majority class, it struggles to accurately segment the minority class.

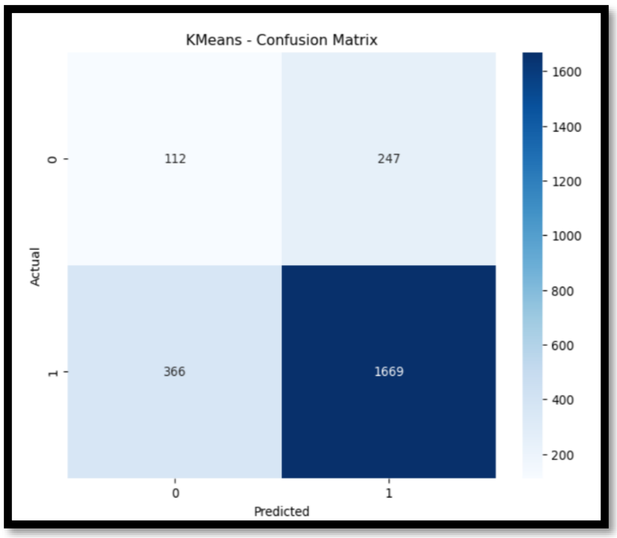
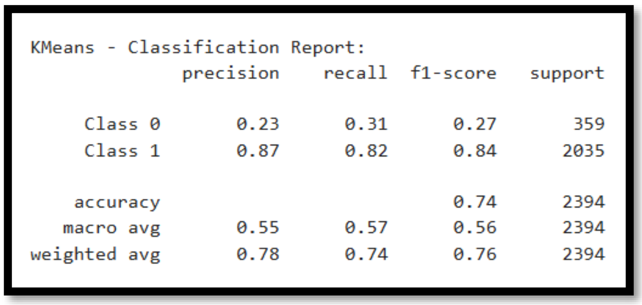


Fig 42: Confusion Matrix Heatmap

Classification Report Insights: The classification report reveals a precision of 23% for Class 0, indicating that when the model predicts this class, it is correct only 23% of the time. This low precision, combined with the low recall for Class 0, suggests that the model has difficulty distinguishing this minority class from the majority.

For Class 1, the precision is much higher at 87%, and the F1 score of 84% reflects the model's strong performance in correctly clustering the majority class.



```
KMeans - Classification Report:
              precision    recall  f1-score   support

     Class 0       0.23      0.31      0.27       359
     Class 1       0.87      0.82      0.84      2035

    accuracy                           0.74      2394
   macro avg       0.55      0.57      0.56      2394
weighted avg       0.78      0.74      0.76      2394
```

Fig 43: Classification Report

Generalization and Model Suitability: The K-Means Clustering model provides useful insights into the structure of the dataset, particularly in grouping similar instances together within the majority class. However, the model's overall performance, as indicated by the wide variance in cross-validation scores and the low precision and recall for the minority class, suggests that there is substantial room for improvement.

This could involve adjusting the number of clusters, fine-tuning the model, or exploring alternative clustering techniques that might better capture the underlying patterns in the data. Despite these limitations, the K-Means model serves as a valuable tool for exploratory data analysis and offers a starting point for identifying patterns within the dataset.

Chapter 7: Conclusion

This dissertation has aimed to bridge the gap between traditional combinatorial methods and modern machine learning approaches in the study of Fano lattice polygons, with a particular focus on predicting the number of T-singularities. By understanding the strengths of both fields, this research has provided valuable insights into the relationship between Ehrhart numbers and T-singularities and evaluated the effectiveness of various machine learning models in this context**.**

7.1 Key Findings:

1. **Logistic Regression:**
    - The model demonstrated strong predictive capabilities with high accuracy and F1 scores, suggesting that it is well-suited for tasks involving binary classification of T-singularities. The model's generalization capabilities were confirmed by consistent performance across different data splits, making it a reliable tool for initial predictive analysis in algebraic geometry.
2. **Random Forest:**
    - While Random Forest showed excellent performance on the training set, its lower test accuracy indicated potential overfitting. However, the model's ability to capture complex interactions within the data was evident, making it a valuable method for understanding the relationships between Ehrhart numbers and T-singularities.
3. **Gradient Boosting:**
    - Gradient Boosting provided a balanced approach with minimal overfitting, offering high accuracy and F1 scores. Its robustness across different data partitions suggests that it is particularly effective for predictive modeling in scenarios involving Fano lattice polygons, especially where precision and recall are equally important.
4. **Support Vector Machine (SVM):**
    - The SVM model exhibited strong performance on the training data but struggled with generalization, particularly in handling class imbalances related to T-singularities. Despite these challenges, SVM's capability in high-dimensional spaces suggests that further refinement could enhance its utility in this domain.
5. **K-Means Clustering:**

- As an unsupervised method, K-Means provided insights into the natural groupings of lattice polygons. However, its performance was less consistent, particularly in accurately identifying the minority class. While useful for exploratory analysis, K-Means may require additional refinement for more reliable predictions in this context.

7.2 Contributions to Algebraic Geometry:

This research contributes to the field of algebraic geometry by demonstrating the applicability of machine learning techniques to the study of Fano lattice polygons. The findings underscore the potential of integrating Ehrhart theory with modern computational methods to predict T-singularities more accurately. Moreover, the study has highlighted the strengths and limitations of different machine learning models, offering a foundation for future research in computational algebraic geometry.

7.3 Limitations and Future Work:

While the models developed in this dissertation have shown promising results, the study is not without limitations. The focus on two-dimensional Fano lattice polygons, while necessary for tractability, limits the generalizability of the findings to higher dimensions. Future research could extend this work to three or higher-dimensional polytopes, which would present additional computational challenges but also offer richer insights.

Additionally, the models' performance on class imbalances suggests that alternative approaches, such as ensemble methods or deep learning techniques, may be necessary to improve accuracy further. Exploration of these advanced methods could yield even more robust tools for predicting T-singularities, with implications not only for algebraic geometry but also for related fields like combinatorial design and theoretical physics.

References:

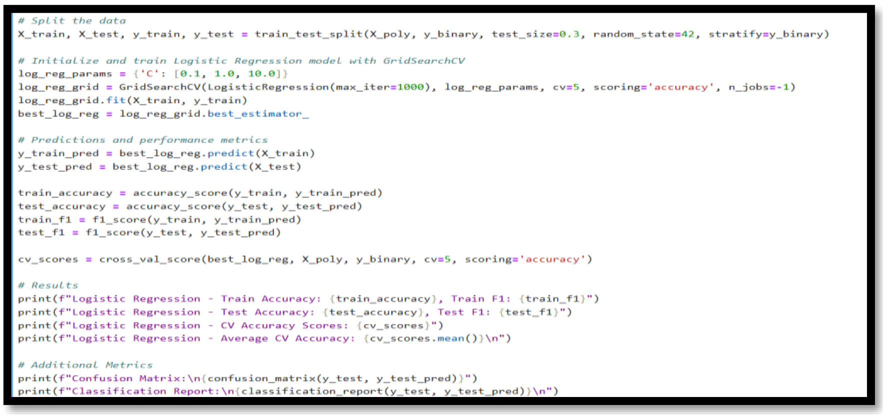
1. Altmann, K., & Hein, G. (2006). A fansy divisor on M¯0,n\bar{M}_0,nM¯0,n. *Journal of Pure and Applied Algebra, 212*(4), 840-850. [A fansy divisor on M¯0,n - ScienceDirect]
2. Batyrev, V. V. (1994). Dual Polyhedra and Mirror Symmetry for Calabi-Yau Hypersurfaces in Toric Varieties. *Journal of Algebraic Geometry, 3*(3), 493-535.[[alg-geom/9310003] Dual Polyhedra and Mirror Symmetry for Calabi-Yau Hypersurfaces in Toric Varieties (arxiv.org)]
3. Batyrev, V. V. (1999). On the Classification of Toric Fano 4-folds. *Journal of Mathematical Sciences, 94*(1), 1021-1050.[[math/9801107] On the Classification of Toric Fano 4-folds (arxiv.org)]
4. Bergstra, J., & Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research, 13*, 281-305.[bergstra12a.dvi (mit.edu)]
5. Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.[Pattern Recognition and Machine Learning | SpringerLink]
6. Breiman, L. (2001). Random Forests. *Machine Learning, 45*(1), 5-32.[Random Forests | Machine Learning (springer.com)]
7. Coates, T., Corti, A., Galkin, S., & Kasprzyk, A. (2016). Quantum periods for 3-dimensional Fano manifolds. *Geometry & Topology, 20*(1), 103-256.[[1303.3288] Quantum Periods for 3-Dimensional Fano Manifolds (arxiv.org)]
8. Cox, D. A., Little, J. B., & Schenck, H. K. (2011). *Toric Varieties*. American Mathematical Society.[David A. Cox, John B. Little, Henry K. Schenck: "Toric Varieties" | Jahresbericht der Deutschen Mathematiker-Vereinigung (springer.com)]
9. Cortes, C., & Vapnik, V. (1995). Support-Vector Networks. *Machine Learning, 20*(3), 273-297.[Support-vector networks | Machine Learning (springer.com)]
10. Davis, J., & Goadrich, M. (2006). The relationship between Precision-Recall and ROC curves. *Proceedings of the 23rd International Conference on Machine Learning (ICML 2006)*, 233-240.[The relationship between Precision-Recall and ROC curves | Proceedings of the 23rd international conference on Machine learning (acm.org)]
11. Dais, D. I. (2002). Resolving 3-dimensional toric singularities. In *Geometry of Toric Varieties* (pp. 155-186). Société mathématique de France.[[math/0110278] Resolving 3-dimensional toric singularities (arxiv.org)]
12. Elisseeff, A., & Weston, J. (2002**).** A kernel method for multi-class classification. *Journal of Machine Learning Research (JMLR)*, 6, 1-36. [http://www.jmlr.org/papers/volume6/elisseeff05a/elisseeff05a.pdf]
13. Fulton, W. (1993). *Introduction to Toric Varieties*. Princeton University Press.[Introduction to Toric Varieties. (AM-131), Volume 131 (degruyter.com)]
14. Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics, 29*(5), 1189-1232.[Greedy function approximation: A gradient boosting machine. (projecteuclid.org)]
15. Gelfand, I. M., Kapranov, M. M., & Zelevinsky, A. V. (1994). *Discriminants, Resultants, and Multidimensional Determinants*. Birkhäuser.[Discriminants, Resultants, and Multidimensional Determinants | SpringerLink]
16. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (2nd ed.). Springer.[The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition | SpringerLink]
17. Hosmer, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). *Applied Logistic Regression* (3rd ed.). Wiley.[Applied Logistic Regression | Wiley Series in Probability and Statistics

18. Hyndman, R. J., & Athanasopoulos, G. (2018). *Forecasting: Principles and practice*. OTexts. [https://otexts.com/fpp3/]
19. Jolliffe, I. T. (2002). *Principal Component Analysis* (2nd ed.). Springer.[Principal Component Analysis | SpringerLink]
20. James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning: With applications in R*. Springer. [https://doi.org/10.1007/978-1-4614-7138-7]
21. Kasprzyk, A. M. (2010). Canonical toric Fano threefolds. *Canadian Journal of Mathematics, 62*(6), 1293-1309.[[0806.2604] Canonical toric Fano threefolds (arxiv.org)
22. Kawamata, Y., Matsuda, K., & Matsuki, K. (1988). Introduction to the Minimal Model Problem. *Advances in Algebraic Geometry*, 1(1), 1-25.[[PDF] Introduction to the Minimal Model Problem | Semantic Scholar]
23. Kohavi, R. (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 2, 1137-1145.[(PDF) A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection (researchgate.net)
24. Little, R. J. A., & Rubin, D. B. (2019). *Statistical Analysis with Missing Data* (3rd ed.). Wiley. [Roderick J. Little and Donald B. Rubin: Statistical Analysis with Missing Data | Psychometrika (springer.com)]
25. MacQueen, J. (1967). Some Methods for Classification and Analysis of Multivariate Observations. *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, 1*, 281-297.[[PDF] Some methods for classification and analysis of multivariate observations | Semantic Scholar]
26. Montufar, G. F., Pascanu, R., Cho, K., & Bengio, Y. (2014). On the Number of Linear Regions of Deep Neural Networks. *Advances in Neural Information Processing Systems, 27*, 2924-2932.[[1402.1869] On the Number of Linear Regions of Deep Neural Networks (arxiv.org)]
27. Mori, S., & Mukai, S. (1981). Classification of Fano 3-folds with B2≥2B_2 \geq 2B2≥2. *Manuscripta Mathematica, 36*(2), 147-162.[Classification of Fano 3-folds with B2≥2 | manuscripta mathematica (springer.com)]
28. Pandas. (n.d.). *Pandas Documentation*. Retrieved from https://pandas.pydata.org/docs/
29. Pick, G. (1899). Geometrisches zur Zahlentheorie. *Journal für die reine und angewandte Mathematik, 133*, 170–176.[Grundlagen der Zahlentheorie | SpringerLink]
30. Python Community. (n.d.). *Python Community*. Retrieved from https://www.python.org/community/
31. Python Software Foundation. (n.d.). *Python Programming Language*. Retrieved from https://www.python.org/
32. Reid, M. (1980). Canonical 3-Folds. In: *Algebraic Geometry and Commutative Algebra*, 15–50.[[PDF] Minimal Models of Canonical 3-Folds | Semantic Scholar]
33. Reid, M. (1985). Young Person's Guide to Canonical Singularities. In S. J. Bloch (Ed.), *Algebraic Geometry Bowdoin 1985* (pp. 345-414). American Mathematical Society.[[PDF] Young person''s guide to canonical singularities | Semantic Scholar]
34. Scikit-learn Documentation. (n.d.). *Confusion matrix*. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html
35. Scikit-learn Documentation. (n.d.). *Cross_val_score*. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html
36. Scikit-learn Documentation. (n.d.). *Evaluation Metrics*. Retrieved from https://scikit-learn.org/stable/modules/model_evaluation.html
37. Scikit-learn Documentation. (n.d.). *Gradient Boosting Classifier*. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

38. Scikit-learn Documentation. (n.d.). *GridSearchCV*. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
39. Scikit-learn Documentation. (n.d.). *KMeans*. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
40. Scikit-learn Documentation. (n.d.). *Logistic Regression*. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
41. Scikit-learn Documentation. (n.d.). *Log Loss*. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.log_loss.html
42. Scikit-learn Documentation. (n.d.). *Polynomial Features*. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html
43. Scikit-learn Documentation. (n.d.). *Principal Component Analysis*. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
44. Scikit-learn Documentation. (n.d.). *Random Forest Classifier*. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
45. Scikit-learn Documentation. (n.d.). *Standard Scaler*. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
46. Scikit-learn Documentation. (n.d.). *Support Vector Classifier*. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
47. Seaborn Documentation. (n.d.). *Seaborn Documentation*. Retrieved from https://seaborn.pydata.org/
48. SQLite Documentation. (n.d.). *SQLite Documentation*. Retrieved from https://www.sqlite.org/docs.html
49. Statsmodels Documentation. (n.d.). *Statsmodels Documentation*. Retrieved from https://www.statsmodels.org/
50. Visual Studio Code. (n.d.). *Visual Studio Code*. Retrieved from https://code.visualstudio.com/
51. Vapnik, V. N. (1998). *Statistical Learning Theory*. Wiley.[Statistical learning theory | Semantic Scholar]
52. Jupyter Notebook. (n.d.). *Jupyter Notebook*. Retrieved from https://jupyter.org/
53. Matplotlib Documentation. (n.d.). *Matplotlib Documentation*. Retrieved from https://matplotlib.org/
54. NumPy Documentation. (n.d.). *NumPy Documentation*. Retrieved from https://numpy.org/doc/stable/
55. Python Software Foundation. (n.d.). *Python Programming Language*. Retrieved from https://www.python.org/

Appendix A: Supplementary Materials I

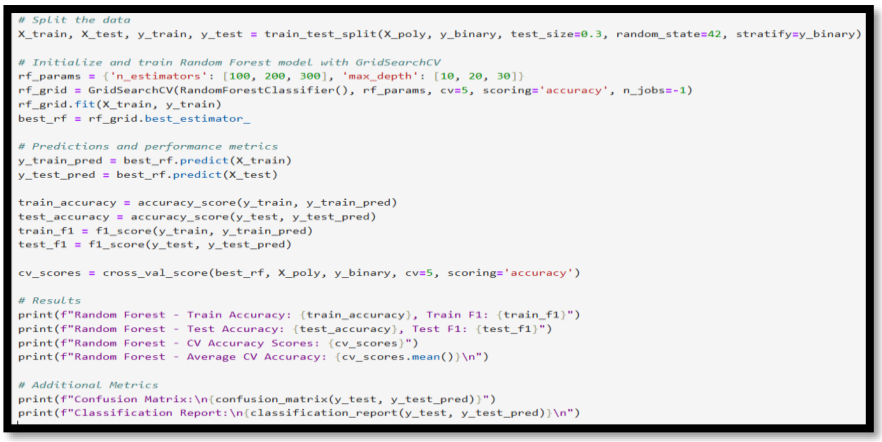
1. Code for Logistic Regression Model:



```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_poly, y_binary, test_size=0.3, random_state=42, stratify=y_binary)

# Initialize and train Logistic Regression model with GridSearchCV
log_reg_params = {'C': [0.1, 1.0, 10.0]}
log_reg_grid = GridSearchCV(LogisticRegression(max_iter=1000), log_reg_params, cv=5, scoring='accuracy', n_jobs=-1)
log_reg_grid.fit(X_train, y_train)
best_log_reg = log_reg_grid.best_estimator_

# Predictions and performance metrics
y_train_pred = best_log_reg.predict(X_train)
y_test_pred = best_log_reg.predict(X_test)

train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
train_f1 = f1_score(y_train, y_train_pred)
test_f1 = f1_score(y_test, y_test_pred)

cv_scores = cross_val_score(best_log_reg, X_poly, y_binary, cv=5, scoring='accuracy')

# Results
print(f"Logistic Regression - Train Accuracy: {train_accuracy}, Train F1: {train_f1}")
print(f"Logistic Regression - Test Accuracy: {test_accuracy}, Test F1: {test_f1}")
print(f"Logistic Regression - CV Accuracy Scores: {cv_scores}")
print(f"Logistic Regression - Average CV Accuracy: {cv_scores.mean()}\n")

# Additional Metrics
print(f"Confusion Matrix:\n{confusion_matrix(y_test, y_test_pred)}")
print(f"Classification Report:\n{classification_report(y_test, y_test_pred)}\n")
```

Fig 44: Code for Logistic Regression Model

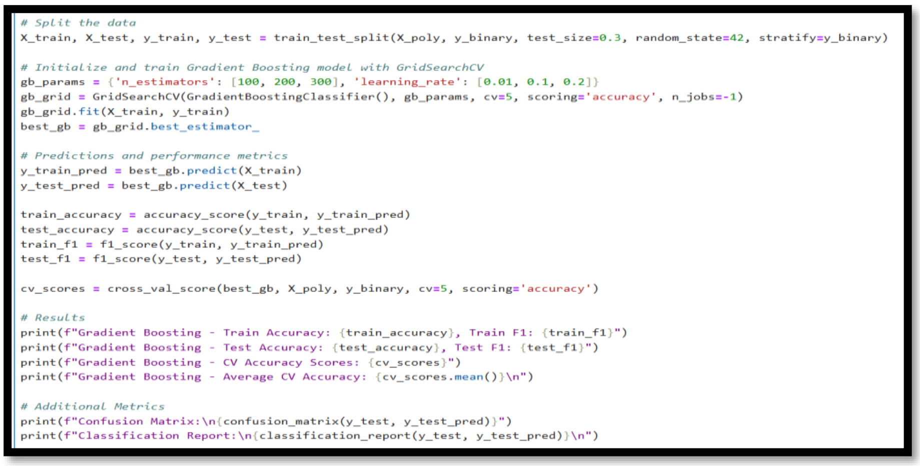
2. Code for Random Forest Model:



```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_poly, y_binary, test_size=0.3, random_state=42, stratify=y_binary)

# Initialize and train Random Forest model with GridSearchCV
rf_params = {'n_estimators': [100, 200, 300], 'max_depth': [10, 20, 30]}
rf_grid = GridSearchCV(RandomForestClassifier(), rf_params, cv=5, scoring='accuracy', n_jobs=-1)
rf_grid.fit(X_train, y_train)
best_rf = rf_grid.best_estimator_

# Predictions and performance metrics
y_train_pred = best_rf.predict(X_train)
y_test_pred = best_rf.predict(X_test)

train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
train_f1 = f1_score(y_train, y_train_pred)
test_f1 = f1_score(y_test, y_test_pred)

cv_scores = cross_val_score(best_rf, X_poly, y_binary, cv=5, scoring='accuracy')

# Results
print(f"Random Forest - Train Accuracy: {train_accuracy}, Train F1: {train_f1}")
print(f"Random Forest - Test Accuracy: {test_accuracy}, Test F1: {test_f1}")
print(f"Random Forest - CV Accuracy Scores: {cv_scores}")
print(f"Random Forest - Average CV Accuracy: {cv_scores.mean()}\n")

# Additional Metrics
print(f"Confusion Matrix:\n{confusion_matrix(y_test, y_test_pred)}")
print(f"Classification Report:\n{classification_report(y_test, y_test_pred)}\n")
```

Fig 45: Code for Random Forest Model

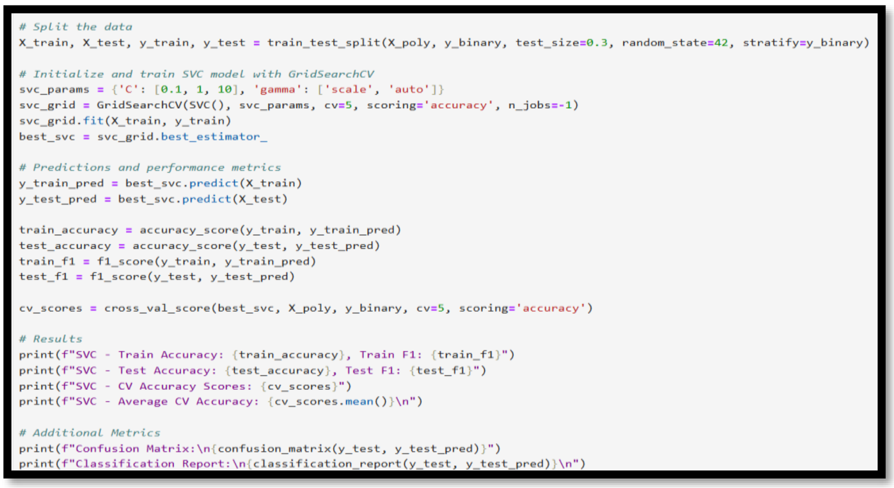
3. Code for Gradient Boosting Model:



```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_poly, y_binary, test_size=0.3, random_state=42, stratify=y_binary)

# Initialize and train Gradient Boosting model with GridSearchCV
gb_params = {'n_estimators': [100, 200, 300], 'learning_rate': [0.01, 0.1, 0.2]}
gb_grid = GridSearchCV(GradientBoostingClassifier(), gb_params, cv=5, scoring='accuracy', n_jobs=-1)
gb_grid.fit(X_train, y_train)
best_gb = gb_grid.best_estimator_

# Predictions and performance metrics
y_train_pred = best_gb.predict(X_train)
y_test_pred = best_gb.predict(X_test)

train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
train_f1 = f1_score(y_train, y_train_pred)
test_f1 = f1_score(y_test, y_test_pred)

cv_scores = cross_val_score(best_gb, X_poly, y_binary, cv=5, scoring='accuracy')

# Results
print(f"Gradient Boosting - Train Accuracy: {train_accuracy}, Train F1: {train_f1}")
print(f"Gradient Boosting - Test Accuracy: {test_accuracy}, Test F1: {test_f1}")
print(f"Gradient Boosting - CV Accuracy Scores: {cv_scores}")
print(f"Gradient Boosting - Average CV Accuracy: {cv_scores.mean()}\n")

# Additional Metrics
print(f"Confusion Matrix:\n{confusion_matrix(y_test, y_test_pred)}")
print(f"Classification Report:\n{classification_report(y_test, y_test_pred)}\n")
```

Fig 46: Code for Gradient Boosting Model

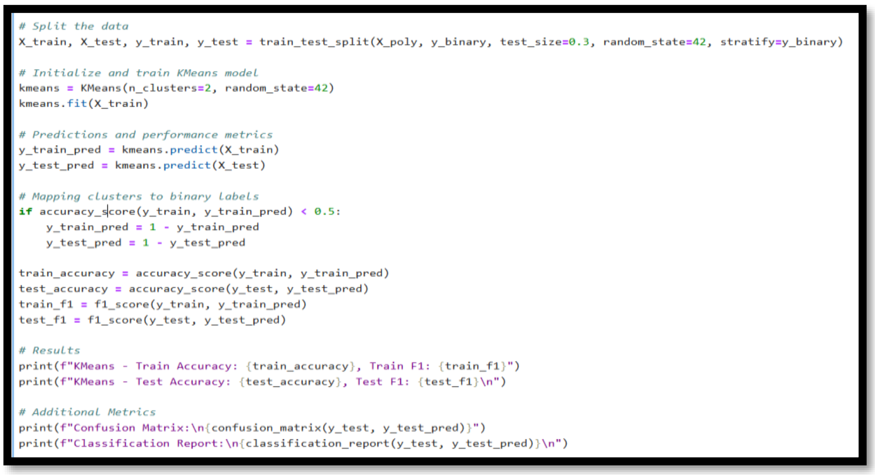
4. Code for SVC Model:



```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_poly, y_binary, test_size=0.3, random_state=42, stratify=y_binary)

# Initialize and train SVC model with GridSearchCV
svc_params = {'C': [0.1, 1, 10], 'gamma': ['scale', 'auto']}
svc_grid = GridSearchCV(SVC(), svc_params, cv=5, scoring='accuracy', n_jobs=-1)
svc_grid.fit(X_train, y_train)
best_svc = svc_grid.best_estimator_

# Predictions and performance metrics
y_train_pred = best_svc.predict(X_train)
y_test_pred = best_svc.predict(X_test)

train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
train_f1 = f1_score(y_train, y_train_pred)
test_f1 = f1_score(y_test, y_test_pred)

cv_scores = cross_val_score(best_svc, X_poly, y_binary, cv=5, scoring='accuracy')

# Results
print(f"SVC - Train Accuracy: {train_accuracy}, Train F1: {train_f1}")
print(f"SVC - Test Accuracy: {test_accuracy}, Test F1: {test_f1}")
print(f"SVC - CV Accuracy Scores: {cv_scores}")
print(f"SVC - Average CV Accuracy: {cv_scores.mean()}\n")

# Additional Metrics
print(f"Confusion Matrix:\n{confusion_matrix(y_test, y_test_pred)}")
print(f"Classification Report:\n{classification_report(y_test, y_test_pred)}\n")
```

Fig 47: Code for SVC Model

5. Code for K-Means Model:



```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_poly, y_binary, test_size=0.3, random_state=42, stratify=y_binary)

# Initialize and train KMeans model
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(X_train)

# Predictions and performance metrics
y_train_pred = kmeans.predict(X_train)
y_test_pred = kmeans.predict(X_test)

# Mapping clusters to binary labels
if accuracy_score(y_train, y_train_pred) < 0.5:
    y_train_pred = 1 - y_train_pred
    y_test_pred = 1 - y_test_pred

train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
train_f1 = f1_score(y_train, y_train_pred)
test_f1 = f1_score(y_test, y_test_pred)

# Results
print(f"KMeans - Train Accuracy: {train_accuracy}, Train F1: {train_f1}")
print(f"KMeans - Test Accuracy: {test_accuracy}, Test F1: {test_f1}\n")

# Additional Metrics
print(f"Confusion Matrix:\n{confusion_matrix(y_test, y_test_pred)}")
print(f"Classification Report:\n{classification_report(y_test, y_test_pred)}\n")
```

Fig 48: Code for SVC Model

6. Graphical Representation of different PCA components:

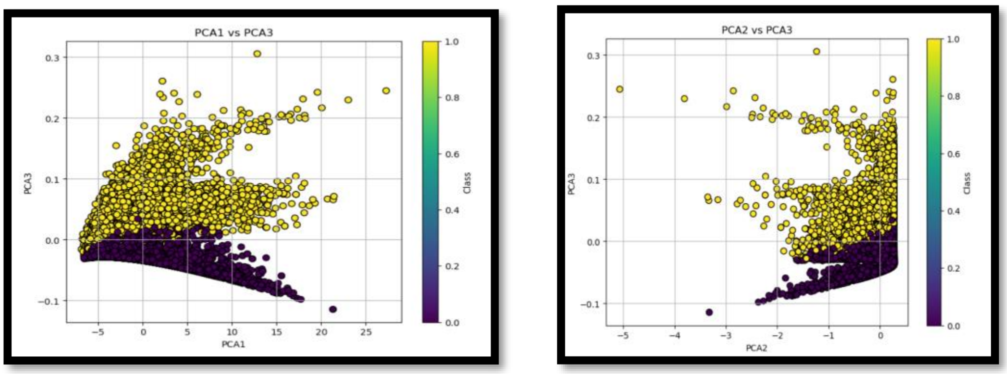
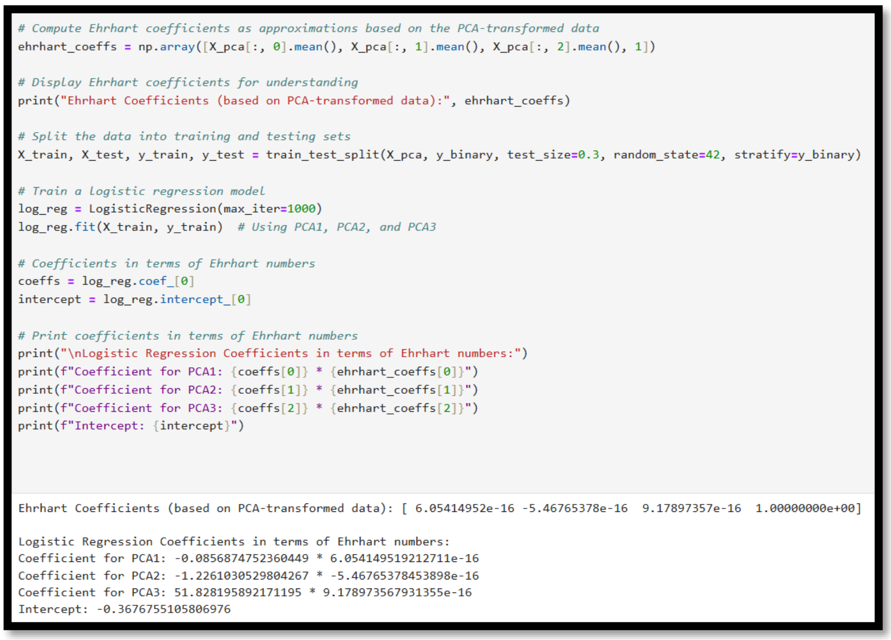


Fig 49: Graphical Representation for PCA Components

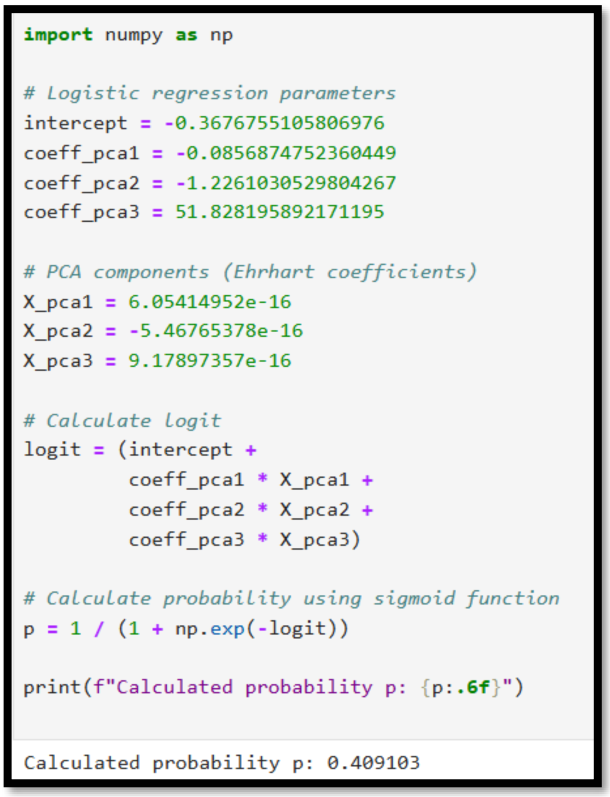
6. Code and Result for coefficient and probability calculation for Logistic Regression Model:



```python
# Compute Ehrhart coefficients as approximations based on the PCA-transformed data
ehrhart_coeffs = np.array([X_pca[:, 0].mean(), X_pca[:, 1].mean(), X_pca[:, 2].mean(), 1])

# Display Ehrhart coefficients for understanding
print("Ehrhart Coefficients (based on PCA-transformed data):", ehrhart_coeffs)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_pca, y_binary, test_size=0.3, random_state=42, stratify=y_binary)

# Train a logistic regression model
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train, y_train)  # Using PCA1, PCA2, and PCA3

# Coefficients in terms of Ehrhart numbers
coeffs = log_reg.coef_[0]
intercept = log_reg.intercept_[0]

# Print coefficients in terms of Ehrhart numbers
print("\nLogistic Regression Coefficients in terms of Ehrhart numbers:")
print(f"Coefficient for PCA1: {coeffs[0]} * {ehrhart_coeffs[0]}")
print(f"Coefficient for PCA2: {coeffs[1]} * {ehrhart_coeffs[1]}")
print(f"Coefficient for PCA3: {coeffs[2]} * {ehrhart_coeffs[2]}")
print(f"Intercept: {intercept}")
```

```
Ehrhart Coefficients (based on PCA-transformed data): [ 6.05414952e-16 -5.46765378e-16  9.17897357e-16  1.00000000e+00]

Logistic Regression Coefficients in terms of Ehrhart numbers:
Coefficient for PCA1: -0.0856874752360449 * 6.054149519212711e-16
Coefficient for PCA2: -1.2261030529804267 * -5.46765378453898e-16
Coefficient for PCA3: 51.828195892171195 * 9.178973567931355e-16
Intercept: -0.3676755105806976
```

Fig 50: Calculation of coefficients in terms of Erhart numbers



```python
import numpy as np

# Logistic regression parameters
intercept = -0.3676755105806976
coeff_pca1 = -0.0856874752360449
coeff_pca2 = -1.2261030529804267
coeff_pca3 = 51.828195892171195

# PCA components (Ehrhart coefficients)
X_pca1 = 6.05414952e-16
X_pca2 = -5.46765378e-16
X_pca3 = 9.17897357e-16

# Calculate logit
logit = (intercept +
         coeff_pca1 * X_pca1 +
         coeff_pca2 * X_pca2 +
         coeff_pca3 * X_pca3)

# Calculate probability using sigmoid function
p = 1 / (1 + np.exp(-logit))

print(f"Calculated probability p: {p:.6f}")
```

```
Calculated probability p: 0.409103
```

Fig 51: Calculation of Probability