Classifiers for Predicting the Success of the Generalization of Fulton's Intersection Multiplicity Algorithm on Polynomial Systems

CS 9637

Taabish Jeshani[1], Wilson Poulter[2], Ryan Sandford[3]

Department of Computer Science, The University of Western Ontario, London, Canada
[1] tjeshan@uwo.ca, [2] wpoulter@uwo.ca, [3] rsandfo@uwo.ca

Abstract. The generalization of Fulton's intersection multiplicity algorithm provides a powerful tool for computing intersection multiplicities without the use of standard bases. Since the generalization of Fulton's algorithm may fail on some inputs it is natural to wonder what properties of the input can be used to determine the algorithm's success before runtime. In this paper we propose several candidate properties of these polynomial input systems and train several classifiers on the features extracted using the proposed properties. The results show that several of the properties can be used to reliably predict the success of the generalization of Fulton's algorithm.

1 Introduction

The study of singularities of algebraic sets gives rise to many important questions in algebraic geometry. When an algebraic set contains a singularity, approximation at that point by a linear space is no longer possible, in which case, tools such as the tangent cone and intersection multiplicity allow one to understand the behaviour of the singularity. Hence, the computation of such tools is an important topic in computer algebra.

The first algorithmic solution to computing intersection multiplicities was given by Mora, which makes use of standard (Gröbner) bases to compute intersection multiplicities, and is described in [2]. Although standard bases are a powerful tool in computer algebra, they face many practical drawbacks, hence, it is reasonable to consider the algorithms for computing intersection multiplicities which do not rely on the computation of standard bases.

Standard basis free intersection multiplicity algorithms often rely on the work of William Fulton, who presented an algorithm for computing intersection multiplicities for planar curves in [3]. Fulton's algorithm applies a set of rewrite rules to simplify a system of two polynomials in two variables until the intersection multiplicity can be computed. Several standard basis free intersection multiplicity algorithms were investigated in [5], [1], [7], which use a reduction criterion to reduce to the bivariate case, for which Fulton's algorithm may be applied. Unfortunately, this criterion does not always hold, particularly, when all polynomials are singular at the point of study.

Recently the authors of [6] investigated a new approach to computing intersection multiplicities without the use of standard bases. Rather than reducing to the bivariate case and applying Fulton's algorithm, the authors extended Fulton's algorithm to a partial algorithm in the n-variate case. This algorithm does not generalize to a complete algorithm, as one of the steps used in Fulton's algorithm does not hold in the generic case. In the 2021 Maple Conference[1], an implementation of this algorithm was presented, as well as bench mark testing which suggests this approach is often

[1] No citation is available as the proceedings have not yet been published.

much faster and can compute more examples than the previous standard basis free intersection multiplicity algorithms. The downside of this generalization of Fulton's algorithm is that it is a partial algorithm, and hence may not always succeed. Moreover, without running the procedure, it is difficult to tell whether a given input system will cause the algorithm to fail, since it is difficult to predict how a system will be rewritten without running the procedure itself.

In this paper we train several classifiers on various properties of polynomial systems which we believe to be related to the success/failure of the generalization of Fulton's algorithm. Moreover, we compare the classifiers and their varying degrees of accuracy. The results suggest that these properties can indeed provide an indication of whether the generalization of Fulton's algorithm will succeed on a given system of polynomial equations.

2 Notation and preliminaries

We abstract much of the mathematical background necessary to understand Fulton's algorithm and it's generalization to the appendices in order to maintain focus on the novel aspects of this paper. In this section, we will provide some conventions used throughout this paper coupled with informal explanations, as to compliment the precise material in the Appendix.

The generalization of Fulton's algorithm is a partial algorithm which returns either the intersection multiplicity of the input or fail if it cannot determine the intersection multiplicity. The input is a system of polynomial equations, a point, and a variable ordering as input. Our goal is to define a set of features such that a classifier can predict whether the generalization of Fulton's algorithm will succeed or fail for a given input system. To do this, we will standardize the training data, enforcing several constraints on our polynomial input systems. A formal definition of Intersection Multiplicity can be found in Appendix A.

2.1 Locality of Intersection Multiplicity

First, we note that intersection multiplicity is a local notion. That is, the intersection multiplicity of a system is defined locally at a given point. Hence, depending on which point we choose to compute the intersection multiplicity at, we may obtain different values. For simplicity, we will study the intersection multiplicity of polynomial systems at the origin.

Training a classifier which only considers intersection multiplicity at the origin is sufficient as to generalize to any point. This is because, to compute the intersection multiplicity at a point which is not the origin, one can apply a linear change of coordinates to the polynomial system, as to assume generically the point of interest is the origin. This property, along with the other properties of Fulton, are stated precisely in Appendix B.

Since we are considering intersection multiplicity at the origin we need only consider systems of polynomial equations which all vanish at the origin. When one polynomial does not vanish at the origin a property of Fulton tells us the intersection multiplicity is zero and the generalization of Fulton's algorithm always returns zero in this case. Hence, the algorithm will succeed trivially. When generating data for our classifier we will therefore ensure all polynomials vanish at the origin, as there is no need to pollute the data with trivial samples.

2.2 Variable Ordering

The generalization of Fulton's algorithm also requires a variable ordering to be specified. Throughout this paper we will assume $x_1 \succ \ldots \succ x_n$. When it is convenient to label variables x, y, z we will

assume $x \succ y \succ z$. Although this variable ordering is necessary for the procedure it is not inherently tied to the notion of intersection multiplicity itself. Hence, it is possible for the generalization of Fulton's algorithm to succeed on one ordering and fail on another. Therefore, when generating data, we will only run the generalization of Fulton's algorithm on one ordering, so our classifiers will only learn on one fixed variable ordering. Once again, by the linear change of coordinates property of intersection multiplicities, we may assume this generically.

2.3 Regular Sequences

The generalization of Fulton's algorithm also requires two additional input constraints. First the system must be square, that is the system has as many polynomial equations as variables. The second constraint is that the polynomial system must form a regular sequence. This constraint is much more technical and is discussed in Appendix A. Informally, this ensures that the intersection multiplicity stays finite during the rewrite process and particularly, during the splitting step.

2.4 Permutations

A permutation, for our purposes here, is a σ function that reorders a set of numbers of the form $\{1, \ldots, n\}$. For example, if $n = 3$, then σ could be given by $\sigma(1) = 3$, $\sigma(2) = 2$, and $\sigma(3) = 1$. This example reverses the normal ordering of 1, 2, 3 to 3, 2, 1. For notation, we let the set of all permutations of $\{1, \ldots, n\}$ be represented by $\mathrm{Sym}(n)$. Note that the size of $\mathrm{Sym}(n)$ is $n!$. Essentially, $\mathrm{Sym}(n)$ captures all the possible arrangements of a set of numbers. This concept is useful when discussing symmetries, as we shall see in §3.4.

2.5 Generating the Data

To generate the training and test data for our classifiers we used Maple′s `randpoly` command. Using this command we narrowed our focus to a subset of polynomial systems, namely that of 3 polynomials in 3 variables with 4 non-zero terms with integer coefficients. We focus on this subset for two main reasons. First, the current implementation of the generalization of Fulton's algorithm is subject to expression swell. Given the limited computing power of the authors, we chose to focus on the smallest size of system for which the algorithm can fail with the hope of reducing the repercussions expression swell brings. Secondly, we fix the number of terms as one of our proposed feature sets examines the number of terms exhibiting a certain property. Allowing systems with varying numbers of non-zero terms could skew this feature. Furthermore, we predict this feature will contribute significantly to our classifiers decision, and hence the number of non-zero terms remains fixed at four.

With Maple′s `randpoly` command we generated over 10 000 polynomial systems of 3 polynomials in 3 variables with 4 non-zero terms, for which all polynomials vanish at the origin. We then used Singular′s `is_regs` command to eliminate systems which were not regular sequences. We then used the implementation of the generalization of Fulton's algorithm from the 2021 Maple conference to classify these systems as successes or failures. Moreover, we discarded any system which took longer than 30 seconds to classify, since the algorithm was likely experiencing severe expression swell in these cases.

This left 2271 systems, with 792 systems successfully computing the intersection multiplicity and 1479 systems for which the algorithm fails. Our data is therefore class imbalanced in favour of the failure class.

3 Methods

In this section, the methods used to model the success/failure of the generalized Fulton's algorithm for a given input in our sample space are discussed. To begin, we outline our problem clearly using mathematical language to outline some properties of our model and for further clarity throughout the rest of the paper. Following, we describe two feature sets that will be used in our classification models. Then, our supervised-learning models and hyper-parameters are discussed. We also discuss the symmetry of our inputs on the function we are modelling, and how we exploit this with what we have called *voting functionals*. The methods used to process the data are summarized. Lastly, our model selection strategy is discussed.

3.1 Analysis of the Problem

In many supervised machine learning problems, it is presupposed that there is some unknown function $f : X \to Y$ and observations $(x, y) \in S \subseteq X \times Y$ such that

$$y = f(x) + \varepsilon,$$

where X is the space of independent features, Y is the space of dependent features, S is the space of observations, and ε is a random variable with $E(\varepsilon) = 0$ that represents noise due to measurement during sampling. The process of supervised learning is to then take some sample $S^* \subseteq S$ and using this sample, construct some model f^* that approximates f through various methods and measures.

While our methodologies in the present paper resemble this general approach to supervised learning, there is a key difference that should be highlighted. Indeed, unlike in the typical scenario described above, our problem does not involve an unknown function. Instead, we are presented with a function given by the generalized Fulton's algorithm, namely $\mathrm{im}_n : X \to \mathbb{N} \cup \{\text{Fail}\}$, where X represent the domain of inputs for the algorithm (see Appendix B). Given this function, we derive a new function $\mathrm{Success}_n : X \to \{0, 1\}$ defined by

$$\mathrm{Success}_n(p; f_1, \ldots, f_n) = \begin{cases} 1 & \text{if } \mathrm{im_n}\,(p;\ f_1, \ldots, f_n) \geq 0 \\ 0 & \text{if } \mathrm{im_n}\,(p;\ f_1, \ldots, f_n) = \text{Fail} \end{cases}$$

Thus, we are interested in producing some function $\mathrm{Success}_n^* : X \to \{0, 1\}$ that approximates $\mathrm{Success}_n$ well, given appropriate measure of approximation for our problem at hand. Of course, because $\mathrm{Success}_n$ is known to us, our problem has a trivial solution given by $\mathrm{Success}_n^* = \mathrm{Success}_n$. Hence we shall insist that our solution $\mathrm{Success}_n^*$ use a different procedure than $\mathrm{Success}_n$, ideally one which can illuminate more precisely why a specific input may succeed or fail. To do so, we rely on feature selection, common supervised learning methods, and additional properties of $\mathrm{Success}_n$ to produce $\mathrm{Success}_n^*$.

3.2 Pre-selection of Features

In this section, we discuss how we transform the data from our sample into two feature sets that will be used for training our models. The purpose of this is to reduce the possibility of over-fitting to features that may have little effect on the outcome of $\mathrm{Success}_n$, and also to make our data compatible with common supervised learning models. Note that this pre-selection does not exclude the possibility that the models, in the process of training, will deselect some of our chosen features due to regularization and hyper-parameter tuning. Described more formally, we produce two functions $\mathrm{Feat}_1 : X \to X^{(1)}$ and $\mathrm{Feat}_2 : X \to X^{(2)}$ where X is the domain of valid inputs

for im_n and $X^{(1)}$ and $X^{(2)}$ are both feature spaces compatible with common supervised learning models. Given our sample $S^* = \{(x_i, y_i) : 0 \leq i \leq m\}$, we produce the new sample spaces $S^{(1)} = \{(\text{Feat}_1(x_i), y_i) : 0 \leq i \leq m\}$ and $S^{(2)} = \{(\text{Feat}_2(x_i), y_i) : 0 \leq i \leq m\}$, which we may train our models on.

Feature Set #1: Coefficients of Polynomial Terms (Naïve Approach) The success or failure of the generalized Fulton's algorithm is determined only by the point p, the variable ordering, and polynomials $f_1, \ldots, f_n$ that are given as inputs. Since, without loss of generality, we may fix the point p and the variable ordering as described in §2.1 and §2.2, we will only study features pertaining to the polynomials $f_1, \ldots, f_n$. One might be interested in selecting $\text{Feat}_1 : X \to X^{(1)}$ such that Feat_1 is bijective. That is, given the equation $y = \text{Feat}_1(x)$ with y given and x unknown, the value of x can be determined uniquely.

The benefit of this property to our model is two-fold. Firstly, we avoid the situation in which inputs x_1 and x_2 respectively succeed and fail on the generalized Fulton's algorithm, but $\text{Feat}_1(x_1) = \text{Feat}_1(x_2)$. For feature sets where this situation arises, our models will *a priori* misclassify some inputs. Secondly, because this feature set completely determines the original input, in some sense we have supplied a locally optimal feature set, since additional features would be redundant, and removing any of these features would likely oversimplify the problem. Note that this is not to say that there could not be other feature sets with the same property but with a simpler feature set.

In the case of our input domain, the most obvious feature set is one that lists the coefficients of the polynomial. Indeed, every polynomial $f(x_1, \ldots, x_n)$ in n variables can be expressed in the form

$$f(x_1, \ldots, x_n) = \sum_{k_1=0}^{N} \cdots \sum_{k_n=0}^{N} \alpha_{(k_1,\ldots,k_n)} x_1^{k_1} \cdots x_n^{k_n}$$

for some sequence $(\alpha_{(k_1,\ldots,k_n)})_0^N$ and some large enough N. Thus, given $(p; f_1, \ldots, f_n)$ in the domain of valid inputs, we can transform it to

$$(p; (\alpha_{(k_1,\ldots,k_n)})_0^N, (\beta_{(k_1,\ldots,k_n)})_0^N, (\gamma_{(k_1,\ldots,k_n)})_0^N))$$

where $(\alpha_{(k_1,\ldots,k_n)})_0^N, (\beta_{(k_1,\ldots,k_n)})_0^N, (\gamma_{(k_1,\ldots,k_n)})_0^N)$ are, respectively, the coefficients of f_1, f_2, f_3.

The major drawback of this feature set is its large dimension. Even if we fix N above so that we only consider polynomials with a maximum multi-degree of $(N, \ldots, N)$, this feature space will have $(N+1)^{n^2} + 1$ dimensions. Indeed, even though a polynomial of the form $2x^2 + xy$ appears to only have the coefficients 2 and 1, we must also include all of its missing coefficients with zeros, i.e., the coefficient of the term x would be 0. Less formally, our input space is a dataframe where every column has a polynomial term as its header. For a given polynomial, we store its coefficients under the matching headers, and put zeros everywhere else.

Hence, many inputs in this feature space are large arrays of mainly zeroes. Due to this, supervised-learning methods be less potent on this feature set as a greater degree of computational power is required. Moreover, these features may not represent the features that are most salient to the problem at hand. Nonetheless, we consider them here as a control feature set for purposes of comparison, despite its naïvety.

Feature Set #2: Number of Vanishing Terms & Modular Degree The key to the success of the generalization of Fulton's algorithm relies on the notion of the modular degree.

Definition 1. *Let f be in $\mathbb{K}[x_1, \ldots, x_n]$ where $x_1 \succ \ldots \succ x_n$. We define the modular degree of f with respect to a variable $v \in V$ as* $\deg_v(f \mod \langle V_{<v} \rangle)$*, where $V = \{x_1, \ldots, x_n\}$ is the set of variables and $V_{<v}$ is the set of all variables less than v in the given ordering. If $V_{<v} = \emptyset$, we define the modular degree of f with respect to v to be the degree of f with respect to v. Write* $\text{moddeg}(f, v)$ *to denote the modular degree of f with respect to v.*

Modular degrees are the key to the rewrite process undergone in the generalization of Fulton's algorithm. Consider the matrix of modular degrees for polynomials $f_1, \ldots, f_n \in \mathbb{K}[x_1, \ldots, x_n]$, such that the i-th,j-th entry of the matrix of modular degrees is the modular degree of f_i with respect to x_j. Simplification occurs in the generalization of Fulton's algorithm when the system of polynomials has a matrix of modular degrees with all entries lying above the anti-diagonal equal to negative infinity. In this case computations split and the problem size is reduced.

Using allowable operations, as to preserve intersection multiplicity, the generalization of Fulton's algorithm seeks to rewrite the system of polynomial equations so that the matrix of modular degrees has the above shape. When this cannot be done, the algorithm returns Fail. Hence, the second feature set will contain features which relate to the algorithm's ability to successfully rewrite the system in this way. This can further be divided into two classes of features relating to the modular degree of the polynomials.

We refer to the first class of features in feature set 2 as the number of vanishing terms. Loosely, the first class tells us how many "bad" terms are contained in a given polynomial. Formally, for a polynomial $f_i \in \mathbb{K}[x_1, \ldots, x_n]$ and a variable x_j such that $j > 1$, the number of vanishing terms is the number of terms t in $f_i \mod \langle x_{j+1}, \ldots, x_n \rangle$ such that the leading coefficient of t with respect to x_j vanishes at the origin.

Example 1. Consider $f \in \mathbb{K}[x, y, z]$, where $x \succ y \succ z$, such that $f = x^3y + x^2y^2 + (x+1)y^3 + xz + (x+2)z^2$. $f \mod \langle z \rangle = x^3y + x^2y^2 + (x+1)y^3$ and hence the number of vanishing terms with respect to y is 2, since both x^3 and x^2 vanish at the origin. Similarly, the number of vanishing terms with respect to z is 1, since there is only one term with positive degree in z whose leading coefficient vanishes at the origin, namely xz.

This may at first seem like a strange feature to study but these vanishing terms are exactly what causes the generalization of Fulton's algorithm to fail (assuming some other conditions are satisfied regarding modular degrees).

The second class of features in feature set 2 is the modular degrees of the polynomials themselves. Loosely, this second class tells us how hard it is for the algorithm to eliminate those "bad" terms described above. Since all vanishing terms with respect to a given variable will have modular degree with respect to that variable bounded by the modular degree of the polynomial, studying the modular degree of the polynomial could prove useful in determining if vanishing terms will be eliminated before causing a failure. Moreover, the polynomial with minimal modular degree is used as a pivot to reduce the modular degrees of other polynomials. If this pivot has a leading coefficient (modulo all variables smaller than that in question) which vanishes at the origin, then a failure will almost always occur.

By studying these two classes of features in the second feature set, which are closely related to the algorithm itself, we predict we will obtain more accurate predictions in comparison to the control set.

3.3 Classification Models and Hyper-parameters

In this section, we list the methods of supervised-learning that were used to model Success_n, along with the various hyper-parameters and optimization strategies that were employed.

Model #1: Logistic Regression The first model that we tested was a logistic regression, which is given by the equation

$$\text{logreg}(\vec{x}) = \frac{1}{1 + e^{\vec{b} \cdot \vec{x}}}$$

where $\vec{x}$ is an input variable, and $\vec{b}$ is a parameter that is optimized to minimize the loss function:

$$\text{Penalty}(\vec{b}) + C \cdot \sum_{i=0}^{N} \log(e^{-y_i(\vec{b} \cdot \vec{x}_i)} + 1).$$

To implement this, the `sklearn.linear_model.LogisticRegression` classifier was used alongside the `SAGA` solver for the optimization problem. In addition to this, the following hyper-parameters were varied over the listed values:

- Penalty: $\{L_1, L_2\}$
- C: $\{1, 0.1, 0.01\}$
- Intercept: {Include, Exclude}

Model #2: Support Vector Machine The second model that we tested was a support vector machine, given by the equation

$$\text{svm}(\vec{x}) = \vec{w} \cdot \varphi_k(\vec{x})$$

where k is a kernel function, φ_k is the transformation given by k (i.e., $k(\vec{x}_i, \vec{x}_j) = \varphi(\vec{x}_i) \cdot \varphi(\vec{x}_j)$), $\vec{x}$ is an input variable, and

$$\vec{w} = \sum_{i=0}^{N} \alpha_i y_i \phi_k(x_i),$$

where $(\alpha_0, \alpha_1 \ldots, \alpha_N)$ is optimized to minimize the constrained expression:

$$\sum_{i=0}^{N} \alpha_i - \frac{1}{2} \sum_{j=0}^{N} \sum_{i=0}^{N} y_i y_j \alpha_i \alpha_j k(\vec{x}_i, \vec{x}_j), \;\; \sum_{i=0}^{N} \alpha_i y_i = 0, \text{ and } 0 \leq \alpha_i \leq C.$$

To implement this, the `sklearn.svm.SVC` classifier was used alongside the `lbfgs` solver for the optimization problem. In addition to this, the following hyper-paramters were varied over the listed values:

- k: {Polynomial, Radial Basis Function, Sigmoid}
- C: $\{1, 0.1, 0.01\}$
- γ: {1, 0.1, 0.01}

Model #3: AdaBoost with Decision Tree The third model that we tested was an AdaBoost classifier with a decision tree used as its weak learner, given by the equation:

$$\text{AdaBoost}(\vec{x}) = \alpha \sum_{i=0}^{M} t_i(\vec{x})$$

where M is the number of estimators, α is the learning rate, $\vec{x}$ is the input vector, and t_{i+1} is a decision tree that trains on a reweighed sample based on t_i's performance, where sample points correctly classified or mis-classified by t_i have decreased or increased weights, respectively.

To implement this, `sklearn.ensemble.AdaBoostClassifier` was used alongside the `SAMME.R` algorithm and the `sklearn.tree.DecisionTreeClassifier` as the weak learner. In addition to this, the following hyper-parameters were varied over the listed values:

- Total Estimators: $\{10, 50, 100, 200\}$
- Learning Rate: $\{1, 0.1, 0.01\}$
- Tree Depth: $\{1, 5, 10\}$

Model #4: Gradient Boosting with Decision Tree The fourth and final model that we tested was a Gradient Boosting classifier with a decision tree used as its weak learner, given by the equation:

$$\text{GradBoost}(\vec{x}) = \alpha \sum_{i=0}^{M} t_i(\vec{x}),$$

where M is the number of estimators, α is the learning rate, $\vec{x}$ is the input vector, and t_{n+1} is a decision tree that is fit the residuals of the training data and the partially boosted classifier $\alpha \sum_{i=0}^{n} t_i(\vec{x})$ according to some loss function.

To implement this, `sklearn.ensemble.GradientBoostingClassifier` was used alongside the `exponential` loss function. In addition to this, the following hyper-parameters were varied over the listed values:

- Total Estimators: $\{10, 50, 100, 200\}$
- Learning Rate: $\{1, 0.1, 0.01\}$
- Tree Depth: $\{5, 6, 7, 8, 9, 10\}$

3.4 Symmetries and Voting in Models

Similar to feature selection, it is useful to take known information about the phenomena that we are modelling and exploit it to our advantage while constructing our models. Notably, $\text{im}_\text{n}(p;\, f_1 \ldots, f_n)$ is *symmetric*, meaning that for any permutation σ of the set $\{1, \ldots, n\}$, the equation

$$\text{im}_\text{n}(p;\, f_1 \ldots, f_n) = \text{im}_\text{n}\big(p;\, f_{\sigma(1)} \ldots, f_{\sigma(n)}\big)$$

holds for all inputs p and $f_1, \ldots, f_n$. Hence, Success_n is symmetric as well.

Given this fact, we might expect that a good model Success_n^* of Success_n would be symmetric as well. In an attempt to do so, we define what we call a *voting functional* as follows:

Definition 2. *Let C be a function $C : X^n \to Y$ and* vote *be a function* $\text{vote} : \mathcal{P}(Y) \to Y$ *(where $\mathcal{P}(Y)$ is the power set of Y). Then, we define the voting functional* vote_n^* *on functions $X^n \to Y$ as:*

$$\text{vote}_n^*(C)(x_1, \ldots, x_n) = \text{vote}(\{C(x_{\sigma(1)}, \ldots, x_{\sigma(n)}) : \sigma \in \text{Sym}(n)\})$$

Note that $\text{vote}_n^*(C)$ is indeed symmetric. Thus, given a suitable voting function, vote, and a trained classifier, C, we may construct a new classifier $C_\text{vote} = \text{vote}_n^*(C)$ which, given our choice of vote, may be an improvement upon C as a model of Success_n.

As candidates for vote in our model, we consider the voting functions max, min, and mean, where each of these take upon their usual interpretation of as a set-valued function. Thus, given a trained binary classifier C:

– $C_{\max}$ assigns class 1 to the inputs if and only if at least one permutation of the inputs is assigned class 1 by the trained classifier C.

– $C_{\min}$ assigns class 1 to the inputs if and only if none of the permutations of the inputs is assigned class 0 by the trained classifier C.

– C_{mean} assigns class 1 to the inputs if and only if the majority of permutations of the inputs are assigned class 1 by the trained classifier C.

As their name suggests, the *voting functionals* create a formalization of a voting process, where a classifier C would place multiple *votes* on the class of an input, based on how it classifies its permutations.

3.5 Data Processing

In this section, we recount the steps that we took to prepare our data for cross-validation, training, and testing, which included the removal of duplicates, the transformation of feature sets, the regularization of features, and the stratified separation of training and test sets.

As mentioned in §2.4, our data was generated according to Maple's `randpoly`, which is a pseudo-random polynomial generator. At times, this sampling method produced duplicate data points, to a total of nearly 380 duplicates. On the other hand, a truly random sampling method from the space of polynomials would produce duplicates with a probability of 0. Due to this, we removed all duplicated data from our data set, as this could only be explained by limited sampling techniques.

After the duplicated data was removed, for each data point in our sample (x_i, y_i), we replaced it with the transformed data point $(\text{Feat}_1(x_i), \text{Feat}_2(x_i), y_i)$. Following this, these data points again transformed by normalization, so that the new point is given by

$$\left(\frac{\text{Feat}_1(x_i) - \text{Feat}_1(x)_\mu}{\text{Feat}_1(x)_\sigma}, \frac{\text{Feat}_2(x_i) - \text{Feat}_2(x)_\mu}{\text{Feat}_2(x)_\sigma}, y_i \right),$$

allowing for regularization during cross-validation.

Lastly, due to a failure-to-success class imbalance of about 2:1, it was decided that when splitting the data into mutually exclusive training and testing sets that the data should be stratified by class. Indeed, although after random sampling we would expect a similar class imbalance in both the training and test sets, we want to ensure that our model is able to train on a significant amount of both classes, and that it can likewise be tested on a significant amount. If we do not stratify our sampling, then this may not be the case. Following this stratification, we split the data into training and test data according to a 7:3 ratio.

3.6 Model Selection and Testing

In this section, we describe the methodology by which we selected our models for testing.

We follow an exhaustive search strategy for cross-validation, where our grid composes all feature sets, classifiers, their accompanying hyper-parameters, and voting methods. The possible feature sets are those described in §3.2, the classifiers and accompanying hyper-parameters are those described in §3.3, and the voting methods are those in §3.4 in addition to the case of applying no voting method whatsoever.

The training data set was divided into two mutually exclusive sets following stratification for the purposes of a 2-fold cross-validation strategy on the set of all potential models. The justification

for this small value is the size of the grid-search, stronger validation methodologies led to significant increases in computational time that were outside the resources of our study.

To conduct cross-validation, each model in our selection (described by the feature set, classifier and hyper-parameters, and voting method selected) was trained on the training data of a given iteration of the 2-fold cross-validation using only the relevant feature set and the classifier and hyper-parameters. Following this training, the trained classifier was affixed with it's voting method. Lastly, the test data of the given iteration of the 2-fold cross-validation was used on this model to produce a set of predictions that were then compared to the actual set of predictions to compute its ROC AUC score. The average of these scores over both iterations of the cross-validation was recorded for each model. A visual summary of this process can be seen in Figure 2 in Appendix C.

Following, the best feature set, hyper-parameters, and voting method was selected for each classifier type based on these ROC AUC scores, leaving four models total for testing. Each of these models was trained on all of the training data as above, and tested against all of the unseen data.

4 Discussion of Results

In this section, the performance of the different classification methods on the same data set is compared and explored. Alongside the classification methods, the results of our exhaustive search for the hyper-parameters for each of the classifiers are analyzed. Specifically highlighting the selected feature set and the voter results for each model. Lastly, it is shown which model performed the best based on selected evaluation metrics.

4.1 Effect of Voting

The results of our exhaustive grid search for determining the best voting functional (see §3.4 for definition) for our classifier was consistent in the sense that the presence of such a voting functional is better than its absence, as all selected models featured some voting functional. That said, our logistic regression model chose the *average* voting functional, while all the remaining models chose *maximum* (see Tables 1-4 of Appendix C under "Voter"). Each of the featured voting functional has its own interpretation below.

Regarding the logistic regression's selection of average, it is notable that the logistic regression model performed better than all other models with respect to recall. This means that the logistic regression had the most true positives of all the models. This is curious because, all else equal, a classifier with an average voting functional should produce less true positives than one with a maximum voting functional. This tells us that the logistic regression was producing more positive values in general than the other classifiers, which is indicated by the number of false positives it has.

With respect to the other models, the selection of the maximum voting functional indicates that the remaining models perform conservatively on unseen data, likely due to class imbalance favoured towards negative cases. The influence of the maximum voter on a conservative classifier is net positive because it will only classify as positive inputs that it has relative certainty for, hence its selection in these classifiers.

4.2 Effect of Feature Sets

The results of our exhaustive grid search for determining the best feature set for each classifier was constant across the board. The second feature set: the number of vanishing terms and modular degree was selected as a hyper-parameter in each top-performing classifier which is observed

in Tables 1-4 of Appendix C under "Feature Set". This suggests that the second feature set is more significant to our classification than the naïve approach. As previously stated in §3.2, our cross-validation search validates our hypothesized importance of the modular degrees to the success/failure of the generalized Fulton's algorithm. Furthermore, including the number of vanishing terms and modular degrees in the the feature set is further validated by our results, as they suggest that it contributes to the failure of the generalization of Fulton's algorithm for a given input. Included in Figure 5 is the relative importance of each feature for the Gradient Boost model. Notably, the modular degree of the polynomials with respect to y is most important, followed by the number of vanishing terms with respect to y. This matches our intuition as the boundary variables x and z are treated differently in the generalization of Fulton's algorithm. This special treatment of the boundary variables is why Fulton's bivariate algorithm does not fail. Hence, it makes sense that properties relating to these variables are given less importance when determining the failure of the generalization of Fulton's algorithm. A notable observation is that the importance of features is roughly symmetric with respect to polynomial inputs. This is also aligned with what we would expect, given the symmetry of Success_n.

4.3 Model Results

Logistic Regression: As Table 1 of Appendix C shows, our exhaustive search determined L_2 penalty was more fitting for our model. This suggests our dataset does not contain many outliers. In general, the L_2 loss function may result in huge deviations when outliers are present and therefore impact the overall accuracy of the classifier. Alongside the penalty, the regularization value was set to 0.1 indicating a strong regularization, meaning more weight is given to the penalty at the expense of fitting to training data.

Support Vector Machine: In Table 2 of Appendix C, the kernel selected for our SVM is the *radial basis function* which is the most widely used kernel for SVMs as it draws similarities to both Gaussian distributions and K-NNs. To go with our RBF kernel we have a gamma value of 0.1 which suggests our classifier requires less curvature/high spread for its decision boundaries.

AdaBoost & Gradient Boost: Tables 3 and 4 of Appendix C go over the hyper-parameters selected for our last two models which have similar parameter results. The number of estimators represents the number of trees used; 100 trees for AdaBoost versus 200 trees for Gradient Boost. The difference in the number of trees can be explained by gradient boost being robust to overfitting; as a result, a larger number of estimators can lead to better performance. Moreover, the learning rate selected for both classifiers is 0.01, suggesting that our model spends more time training to converge to a local minima; in other words, arriving at the best accuracy.

Table 5 of Appendix C highlights the evaluation scores for all the models. With our findings, logistic regression was determined to be the worst-performing classifier and is visibly displayed by our results; scoring the lowest in precision (0.471810), F-measure (0.553043), and the main evaluation metric ROC AUC (0.634034). The Adaboost with the weak tree learner was our second worst performing model noticeably scoring the lowest in classification accuracy (0.518771). Followed by the support vector machine which had comparable results to the Adaboost in ROC AUC: 0.682542 versus 0.660901 respectively. Finally, our gradient boost achieved the highest scores in accuracy (0.737921), precision (0.641148), F-measure (0.599553), and ROC AUC (0.697243). Hence, resulting in our gradient boost classifier being the best performing model for our use case.

5 Future Work

Although the results of this study have been suggestive of the number of vanishing terms and the modular degree of terms as being a significant determinant of the success/failure of the generalized Fulton's algorithm, a number of questions present themselves due to the limitations of the present work.

The sample spaced used in this study is restricted to only polynomials with four terms, three variables, and integer coefficients. Future work could continue to test our hypothesis here by determining the effectiveness of this feature set on inputs with a greater number of terms and polynomials. To make this realizable with the resources at our disposal, we may generate a base set of regular sequences of polynomials in different numbers of variables with varied numbers of terms, and then take advantages of properties of regular sequences to generate more data without relying on Maple's `randpoly`.

The scores of accuracy of our selected model, while suggestive of non-random classification occurring along this feature set, are not strong. In future work, with a larger pool of data, it would be useful to determine how our models improve with respect to more training data. Testing if the behaviour of different measures such as accuracy, F-measure, and ROC AUC, etc., are asymptotic with respect to the size of training sets would be useful here.

Our use of exhaustive search limited our method's accuracy of validation and our ability to search for a wider and more precise set of hyper-parameters. Future works could employ more sophisticated hyper-parameter selection techniques that would allow for a wider range of values to be tested, while also allowing for validation with a greater number of iterations on the training set.

Lastly, continuing to investigate the properties of the generalized Fulton's algorithm would be useful to the improvement of our models, as has been shown here with our cross-validation's selection of the number of vanishing terms and modular degree feature set and a non-trivial voting method in each model. The relative importance of the features given in Figure 5 raises several interesting questions for future work regarding how the pattern observed generalizes. It would be interesting to study how the importance of the number of vanishing terms relative to the importance of the corresponding modular degree changes, once the constraint of only 4 terms is loosened. It is also clear that the modular degree and number of vanishing terms with respect to the middle variable are given more importance than the first and last variables. It's reasonable to assume this pattern would hold for systems of n equations in n variables, since the first and last variable are treated specially in the generalization of Fulton's algorithm. More work remains to be done to study the relative importance of middle variables in gradient boost model. For example, for a system of size n, are features relating to x_2 given more weight than features relating to x_{n-1}? Or perhaps all features relating to middle variables are weighted equally. These questions could have interesting consequences in developing blueprint of the ideal variable ordering for the generalization of Fulton's algorithm.

Appendix A Notation and Definitions

Let $\mathbb{K}$ be an algebraically closed field. Let $\mathbb{A}^n$ denote $\mathbb{A}^n(\mathbb{K})$, the affine space of dimension n over $\mathbb{K}$. We define the degree of the zero polynomial to be $-\infty$ with respect to any variable. If I is an ideal of $\mathbb{K}[x_1, \ldots, x_n]$, we denote by $\mathbf{V}(I)$ the algebraic set (aka variety) consisting of the common zeros to all polynomials in I. Let $p \in \mathbb{A}^n$ be a point. An algebraic set $V \subseteq \mathbb{A}^n$ is zero-dimensional if it contains only finitely many points of $\mathbb{A}^n$.

Definition 3 (Local Ring). *We define the local ring at p as*

$$\mathcal{O}_{\mathbb{A}^n,p} := \left\{ \frac{f}{g} \mid f, g \in \mathbb{K}[x_1, \ldots, x_n] \text{ where } g(p) \neq 0 \right\}.$$

Local rings have a unique maximal ideal. For $\mathcal{O}_{\mathbb{A}^n,p}$, all elements which vanish on p are in the maximal ideal and all of those that do not are units. Hence, given an element $f \in \mathbb{K}[x_1, \ldots, x_n]$ we can test whether f is invertible in $\mathcal{O}_{\mathbb{A}^n,p}$ by testing $f(p) \neq 0$.

Definition 4 (Intersection Multiplicity). *Let $f_1, \ldots, f_n \in \mathbb{K}[x_1, \ldots, x_n]$. We define the intersection multiplicity of $f_1, \ldots, f_n$ at $p \in \mathbb{A}^n$ as the dimension of the local ring at p modulo the ideal generated by $f_1, \ldots, f_n$ in the local ring at p, as a vector space over $\mathbb{K}$. That is,*

$$\mathrm{Im}(p;\, f_1, \ldots, f_n) := \dim_{\mathbb{K}}(\mathcal{O}_{\mathbb{A}^n,p} / \langle f_1, \ldots, f_n \rangle).$$

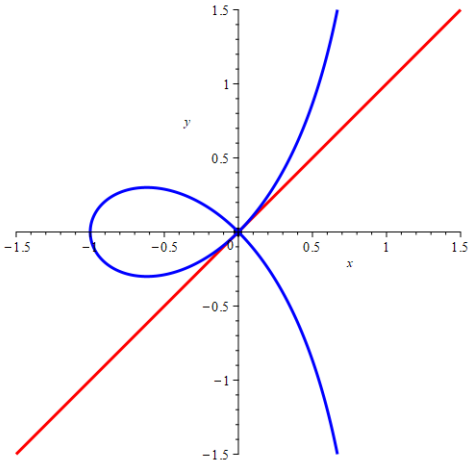


Figure 1: $\mathrm{Im}\big((0,0);\, y - x, x^3 + xy^2 + x^2 - y^2\big) = 3$

An important tool used in the extension of Fulton's algorithm is that of the regular sequence. The authors of [6] require the input system of polynomials form a regular sequence in the local ring at the point of interest. Regular sequences in Noetherian local rings enjoy several nice properties described in [4, Section 3-1], namely, in such a case, being a regular sequence is invariant under permutation. In light of this, we give a simplified definition of a regular sequence, since all regular sequences discussed in this paper will satisfy the above constraint.

Definition 5 (Regular Sequence). *When $f_1, \ldots, f_n \in \mathbb{K}[x_1, \ldots, x_n]$ generate a proper ideal in the local ring at p, we say $f_1, \ldots, f_n$ is a regular sequence if no f_i is zero or a zero divisor modulo $\langle f_1, \ldots, \widehat{f_i}, \ldots, f_n \rangle$, where $\widehat{f_i}$ denotes the omission of f_i.*

Often, we will simply say $f_1, \ldots, f_n$ is a regular sequence when the point p is implicit.

Appendix B Fulton's Algorithm

The following generalization of Fulton's properties was known to the authors of [5] and proved in [7]. These properties extend the properties used by Fulton to characterize intersection multiplicity in the bivariate case.

Theorem 1. *Let $f_1, \ldots, f_n$ be polynomials in $\mathbb{K}[x_1, \ldots, x_n]$ such that $\mathbf{V}(f_1, \ldots f_n)$ is zero-dimensional. Let $p = (p_1, \ldots, p_n) \in \mathbb{A}^n$. Then, the intersection multiplicity $\mathrm{Im}(p;\, f_1, \ldots, f_n)$ satisfies (n-1) to (n-7) where:*

(n-1) $\mathrm{Im}(p;\, f_1, \ldots, f_n)$ *is a non-negative integer.*

(n-2) $\mathrm{Im}(p;\, f_1, \ldots, f_n) = 0$ *if and only if* $p \notin \mathbf{V}(f_1, \ldots, f_n)$.

(n-3) $\mathrm{Im}(p;\, f_1, \ldots, f_n)$ *is invariant under affine changes of coordinates on* $\mathbb{A}^n$.

(n-4) $\mathrm{Im}(p;\, f_1, \ldots, f_n) = \mathrm{Im}(p;\, \sigma(f_1, \ldots, f_n))$ *where σ is any permutation.*

(n-5) $\mathrm{Im}(p;\, (x_1 - p_1)^{m_1}, \ldots, (x_n - p_n)^{m_n}) = m_1 \cdots m_n$ *for any* $m_1, \ldots, m_n \in \mathbb{N}$.

(n-6) $\mathrm{Im}(p;\, f_1, \ldots, f_{n-1}, gh) = \mathrm{Im}(p;\, f_1, \ldots, f_{n-1}, g) + \mathrm{Im}(p;\, f_1, \ldots, f_{n-1}, h)$ *for any* $g, h \in \mathbb{K}[x_1, \ldots, x_n]$ *such that $f_1, \ldots, f_{n-1}, gh$ is a regular sequence in $\mathcal{O}_{\mathbb{A}^n, p}$.*

(n-7) $\mathrm{Im}(p;\, f_1, \ldots, f_n) = \mathrm{Im}(p;\, f_1, \ldots, f_{n-1}, f_n + g)$ *for any* $g \in \langle f_1, \ldots, f_{n-1} \rangle$.

A key notion needed to generalize Fulton's algorithm is that of the modular degree. Modular degrees are used to determine which step of the algorithm to perform. Namely if we define the matrix of modular degrees of $f_1, \ldots, f_n \in \mathbb{K}[x_1, \ldots, x_n]$ to be the matrix whose i-th,j-th entry is the modular degree of f_i with respect to x_j, than the goal of the generalization of Fulton's algorithm is to rewrite the matrix of modular degrees, in a way that preserves intersection multiplicity, so that all entries above the anti-diagonal are negative infinity.

Definition 6. *Let f be in $\mathbb{K}[x_1, \ldots, x_n]$ where $x_1 \succ \ldots \succ x_n$. We define the modular degree of f with respect to a variable $v \in V$ as $\deg_v(f \mod \langle V_{<v} \rangle)$, where $V = \{x_1, \ldots, x_n\}$ is the set of variables and $V_{<v}$ is the set of all variables less than v in the given ordering. If $V_{<v} = \emptyset$, we define the modular degree of f with respect to v to be the degree of f with respect to v. Write* $\mathrm{moddeg}(f, v)$ *to denote the modular degree of f with respect to v.*

The following algorithm generalizes Fulton's algorithm to the n-variate setting. A full description can be found in [6]. The colours are used to provide the reader with a sense of which properties are utilized in the various steps of the algorithm.

Algorithm 1: Generalized Fulton's Algorithm

Function $\mathrm{im_n}(p; f_1, \ldots, f_n)$

Input: Let: $x_1 \succ \ldots \succ x_n$.

1. $p \in \mathbb{A}^n$ the origin.
2. $f_1, \ldots, f_n \in \mathbb{K}[x_1, \ldots, x_n]$ such that $f_1, \ldots, f_n$ form a regular sequence in $\mathcal{O}_{\mathbb{A}^n,p}$ or one such f_i is a unit in $\mathcal{O}_{\mathbb{A}^n,p}$.

Output: $\mathrm{Im}(p; f_1, \ldots, f_n)$ or Fail

if $f_i(p) \neq 0$ *for any i=1,. . . ,n* **then** /* Red */
  return 0
if $n = 1$ **then** /* Compute multiplicity */
  return $\max(m \in \mathbb{Z}^+ \mid f_n \equiv 0 \mod \langle x_1^m \rangle)$
for $i = 1, \ldots, n$ **do**
  for $j = 1, \ldots, n-1$ **do**
    $r_j^{(i)} := \mathrm{moddeg}(f_i, x_j)$
for $j = 1, \ldots, n-1$ **do** /* Orange */
  Reorder $f_1, \ldots, f_{n-j+1}$ so that $r_j^{(1)} \leq \ldots \leq r_j^{(n-j+1)}$ /* Green */
  $m := \min(i \mid r_j^{(i)} > 0)$ or $m \leftarrow \infty$ if no such i exists
  if $m \leq (n-j)$ **then**
    for $i = m+1, \ldots, n-j+1$ **do** /* Blue */
      $d := r_j^{(i)} - r_j^{(m)}$
      $L_m := \mathrm{lc}(f_m(x_1, \ldots, x_j, 0, \ldots, 0); x_j)$
      $L_i := \mathrm{lc}(f_i(x_1, \ldots, x_j, 0, \ldots, 0); x_j)$
      if $L_m(p) \neq 0$ **then**
        $f_i' := L_m f_i - x_j^d L_i f_m$
      else if $L_m \mid L_i$ **then**
        $f_i' := f_i - x_j^d \frac{L_i}{L_m} f_m$
      else
        return Fail
    return $\mathrm{im_n}(p; f_1, \ldots, f_m, f_{m+1}', \ldots, f_{n-j+1}', \ldots, f_n)$
/* Yellow */
for $i = 1, \ldots, n-1$ **do**
  $q_i := \mathrm{quo}(f_i(x_1, \ldots, x_{n-i+1}, 0, \ldots, 0), x_{n-i+1}; x_{n-i+1})$
return
$\mathrm{im_n}(p; q_1, f_2, \ldots, f_n)$
$+ \mathrm{im_{n-1}}(p; q_2(x_1, \ldots, x_{n-1}, 0), \ldots, f_n(x_1, \ldots, x_{n-1}, 0))$
$+$
$\vdots$
$+\mathrm{im_2}(p; q_{n-1}(x_1, x_2, 0, \ldots, 0), f_n(x_1, x_2, 0, \ldots, 0))$
$+\mathrm{im_1}(p; f_n(x_1, 0, \ldots, 0))$

The following example illustrates the rewriting process, and in particular, how it is used to obtain a matrix of modular degrees with a triangular shape. Note that the last column in the matrix of modular degrees, corresponding to the variable x_n, is not computed by the generalization

of Fulton's algorithm, since there are no possible entries above the anti diagonal to rewrite. Hence, the below matrices of modular degrees are $n \times (n-1)$ matrices.

Example 2. Let $f_1, f_2, f_3 \in \mathbb{K}[x, y, z]$ be given by $f_1 = x^2, f_2 = (x+1)y + x^3, f_3 = y^2 + z + x^3$. Suppose we wish to compute the intersection multiplicity at p, the origin. The matrix of modular degrees computed in lines 6-8 of algorithm 1 is:

$$r = \begin{bmatrix} 2 & 0 \\ 3 & 1 \\ 3 & 2 \end{bmatrix},$$

where the i-th row corresponds to the polynomial f_i and the j-th column corresponds to the variable x_j. Hence, (i, j)-th entry is the modular degree of f_i with respect to x_j. Note that we omit the last column which corresponds to the modular degrees with respect to z, as this column has no effect on the conditions necessary to apply Lemma 1 of [6], as to split computations.

Since f_1 has minimal modular degree with respect to x, we choose f_1 as a pivot and use it to reduce the modular degrees of f_2 and f_3 with respect to x. Write

$$f_2' := f_2 - xf_1 = (x+1)y + x^3 - x^3 = (x+1)y,$$

and

$$f_3' := f_3 - xf_1 = y^2 + z + x^3 - x^3 = y^2 + z.$$

By (n-7) we may redefine $f_2 := f_2'$ and $f_3 := f_3'$. Hence, we consider the system given by $f_1 = x^2, f_2 = (x+1)y, f_3 = y^2 + z$. The matrix of modular degrees is now:

$$r = \begin{bmatrix} 2 & 0 \\ -\infty & 1 \\ -\infty & 2 \end{bmatrix},$$

and after reordering f_1, f_2, f_3 by modular degree with respect to x, we have $f_1 = (x+1)y, f_2 = y^2 + z, f_3 = x^2$, with matrix of modular degrees:

$$r = \begin{bmatrix} -\infty & 1 \\ -\infty & 2 \\ 2 & 0 \end{bmatrix}.$$

Since f_1 has minimal modular degree with respect to y, and moreover the leading coefficient of $f_1 \mod \langle z \rangle$ is $x+1$ which is invertible in the local ring at the origin, we may choose f_1 as a pivot and use it to reduce the modular degree of f_2 with respect to y. Write

$$f_2' := (x+1)f_2 - yf_1 = (x+1)y^2 + (x+1)z - (x+1)y^2 = (x+1)z.$$

Redefining $f_2 := f_2'$ and reordering by modular degree in y, we have $f_1 = (x+1)z, f_2 = (x+1)y, f_3 = x^2$, and the matrix of modular degrees is now:

$$r = \begin{bmatrix} -\infty & -\infty \\ -\infty & 1 \\ 2 & 0 \end{bmatrix}.$$

By applying Lemma 1 of [6] and splitting computations we conclude:

$$\begin{aligned} &\mathrm{Im}(p;\, f_1, f_2, f_3) \\ &= \mathrm{Im}(p;\, x+1, f_2, f_3) + \mathrm{Im}(p;\, z, f_2, f_3) \\ &= \mathrm{Im}(p;\, x+1, f_2, f_3) + \mathrm{Im}(p;\, z, x+1, f_3) + \mathrm{Im}(p;\, z, y, f_3) \\ &= 0 + 0 + 2. \end{aligned}$$

Appendix C Tables and Figures

Penalty	Regularization	Intercept	Feature Set	Voter	CV Score
L_2	0.01	Excluded	No. Terms + Mod. Degree	Average	0.66917888

Table 1: Logistic Regression Cross-Validation Results

Kernel	Regularization	Gamma	Feature Set	Voter	CV Score
Radial Basis Function	1	0.1	No. Terms + Mod. Degree	Maximum	0.681360

Table 2: Support Vector Machine Cross-Validation Results

Total Estimators	Learning Rate	Tree Depth	Feature Set	Voter	CV Score
100	0.01	5	No. Terms + Mod. Degree	Maximum	0.691113

Table 3: AdaBoost Cross-Validation Results

Total Estimators	Learning Rate	Tree Depth	Feature Set	Voter	CV Score
200	0.01	5	No. Terms + Mod. Degree	Maximum	0.705638

Table 4: Gradient Boost Cross-Validation Results

Classifier	Accuracy	Precision	Recall	F-Measure	ROC AUC
logreg	0.623719	0.471810	0.668067	0.553043	0.634034
svm	0.559701	0.592885	0.630252	0.592885	0.682542
adaboost_tree	0.518771	0.572505	0.638655	0.572505	0.660901
xgboost	0.737921	0.641148	0.563025	0.599553	0.697243

Table 5: Summary of the test results, with green and red denoting the best and worst results, respectively.

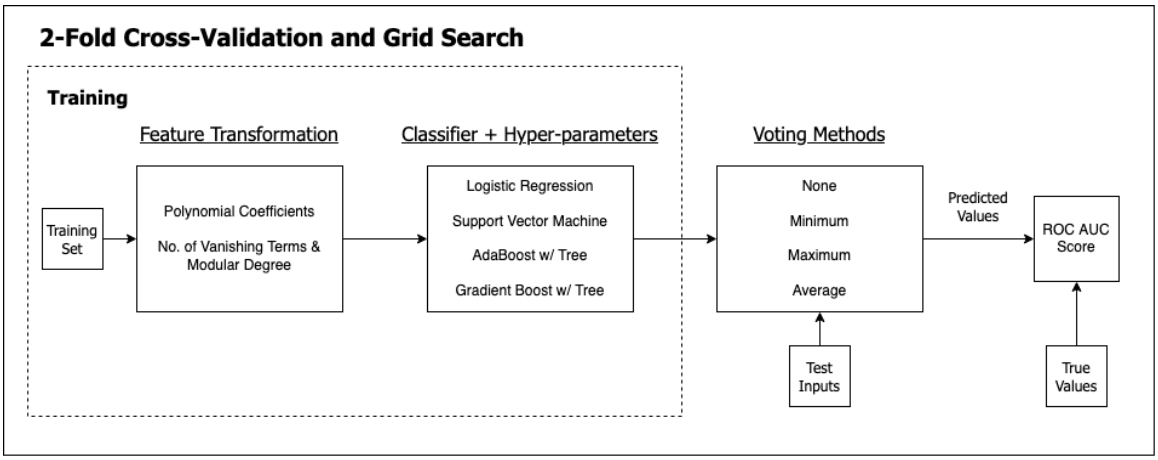
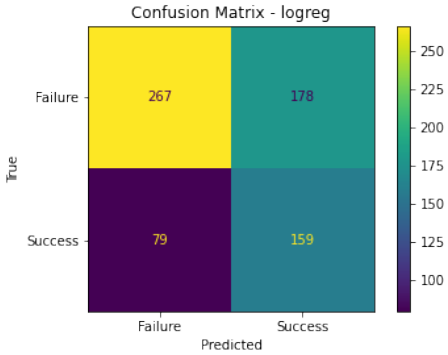
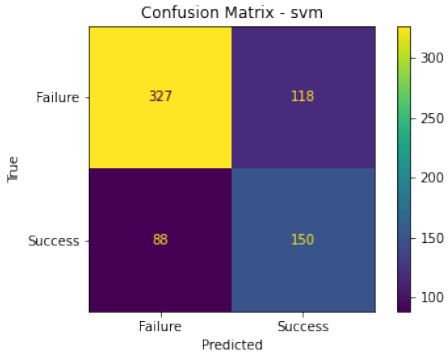


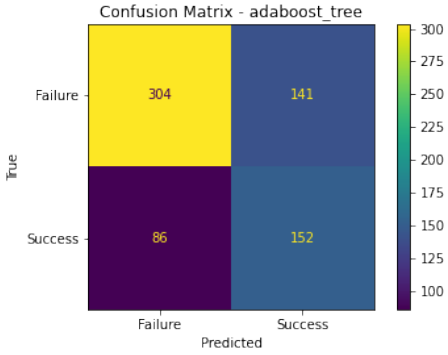
Figure 2: A visualization of our model selection



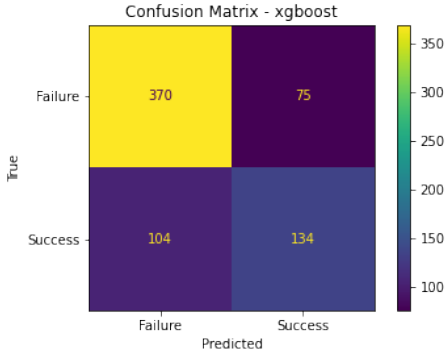
(a) Logistic Regression



(b) Support Vector Machine



(c) AdaBoost w/ Tree



(d) Gradient Boosting w/ Tree

Figure 3: The confusion matrices of all the optimized classifiers.

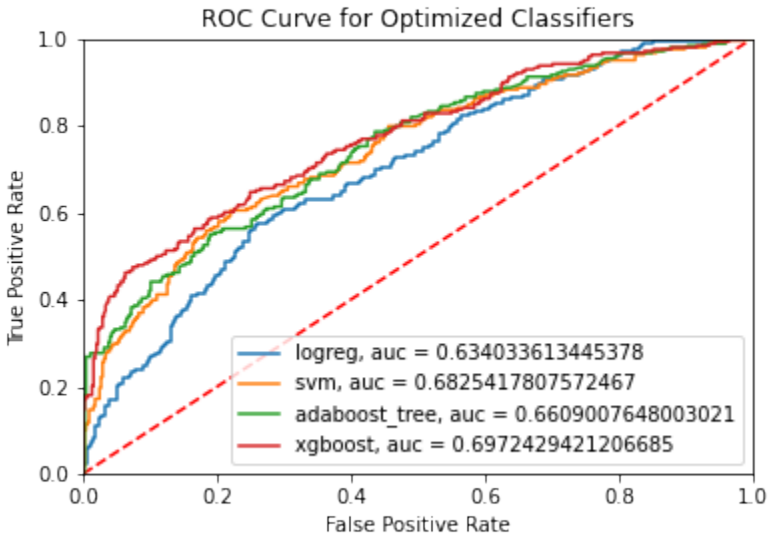


Figure 4: A plot of the ROCs of all the optimized classifiers

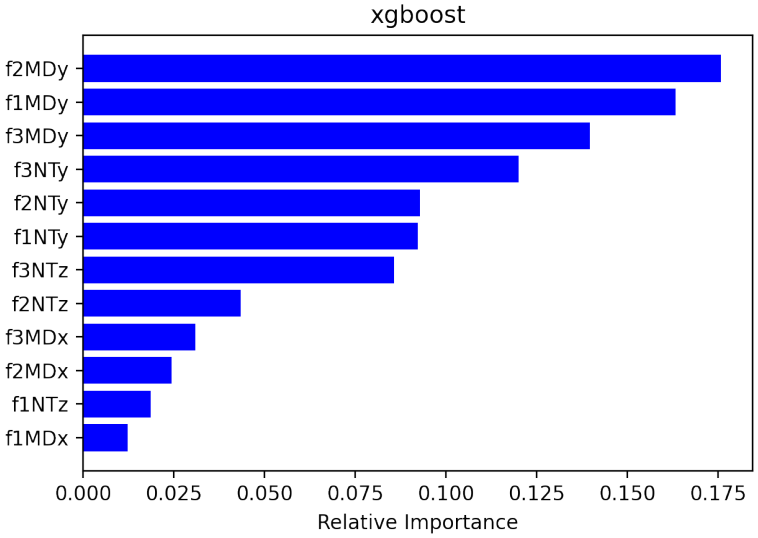


Figure 5: The feature importance for the Gradient Boost model. Note that our labelling convention above is as follow: f1, f2, and f3 represents the polynomial in question, NT and MD represent number of vanishing terms the modular degree respectively, and x, y, and z represent what variable the number of vanishing terms or the modular degree are with respect to. E.g., f2MDy is the modular degree with respect to y of the second input polynomial.

References

1. Parisa Alvandi, Marc Moreno Maza, Éric Schost, and Paul Vrbik. A standard basis free algorithm for computing the tangent cones of a space curve. In Vladimir P. Gerdt, Wolfram Koepf, Werner M. Seiler, and Evgenii V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing*, pages 45–60, Cham, 2015. Springer International Publishing.
2. Wolfram Decker, Gert-Martin Greuel, Gerhard Pfister, and Hans Schönemann. SINGULAR 4-1-1 — A computer algebra system for polynomial computations. `http://www.singular.uni-kl.de`, 2018.
3. William Fulton. *Algebraic curves - an introduction to algebraic geometry (reprint vrom 1969)*. Advanced book classics. Addison-Wesley, 1989.
4. Irving Kaplansky. *Commutative rings*. The University of Chicago Press, Chicago, Ill.-London, revised edition, 1974.
5. Steffen Marcus, Marc Moreno Maza, and Paul Vrbik. On fulton's algorithm for computing intersection multiplicities. In Vladimir P. Gerdt, Wolfram Koepf, Ernst W. Mayr, and Evgenii V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing - 14th International Workshop, CASC 2012, Maribor, Slovenia, September 3-6, 2012. Proceedings*, volume 7442 of *Lecture Notes in Computer Science*, pages 198–211. Springer, 2012.
6. Marc Moreno Maza and Ryan Sandford. Towards extending fulton's algorithm for computing intersection multiplicities beyond the bivariate case. In François Boulier, Matthew England, Timur M. Sadykov, and Evgenii V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing - 23rd International Workshop, CASC 2021, Sochi, Russia, September 13-17, 2021, Proceedings*, volume 12865 of *Lecture Notes in Computer Science*, pages 232–251. Springer, 2021.
7. P. Vrbik. *Computing Intersection Multiplicity via Triangular Decomposition*. PhD thesis, The University of Western Ontario, 2014.