Departamento de Matemáticas, Facultad de Ciencias
Universidad Autónoma de Madrid

Bases de Gröbner y Machine Learning

Trabajo de fin de grado

Grado en Matemáticas

Autor: Santiago González- Carvajal Centenera
Tutor: Dra. María Ángeles Zurro

Curso 2019-2020.

Resumen

La memoria recoge en primer lugar un estudio básico de distintos órdenes monomiales sobre anillos de polinomios. Se analiza la correspondencia entre el Álgebra y la Geometría, básica para entender la interpretación geométrica del conjunto de soluciones de un sitema de ecuaciones polinónicas. Se demuestran dos teoremas de Hilbert, el Teorema de la base y el Nulstellensatz. En la segunda parte de este trabajo, se aplica lo anterior a un experimento usando "Machine Learning": la *rentabilidad* de calcular una base de Gröbner para ideales de polinomios de dos variables y grados arbitrarios. El programa usado ha sido SageMath con unas ampliaciones asociadas a "Machine Learning".

Abstract

The memory contains first a basic study of distinct monomial orders on polynomial rings. The correspondence between Algebra and Geometry is analyzed, basic to understand the geometric interpretation of the set of solutions of a system of polynomial equations. Two Hilbert theorems are proved, the basis Theorem and Nulstellensatz Theorem. In the second part of this work, "Machine Learning" is used to perform an experiment: the *profitability* of computing a Gröbner basis for polynomial ideals in two variables and arbitrary degrees. The program used was SageMath with some extensions associated with "Machine Learning".

Índice general

Introducción

La motivación general del proyecto es estudiar la resolución de sistemas de ecuaciones polinomiales con coeficientes en un cuerpo k, tanto desde el punto de vista teórico como desde un punto de vista más computacional.

La memoria comienza con el estudio de distintos órdenes monomiales sobre anillos de polinomios. Hemos seguido para ello las referencias [8, 9]. Este estudio nos permite introducir el concepto de base de Gröbner de un ideal I de $k[x_1, \ldots, x_n]$ como un sistema de generadores G de I que verifica

$$LT(I) = \langle LT(g) \mid g \in G \rangle \ ,$$

con $LT(f)$ el monomio de mayor grado para el orden monomial fijado. Es un resultado básico del capítulo 1 el llamado Lema de Dickson (teorema 1.6).

Computacionalmente, un base de Gröbner G se calcula usando el algoritmo de Buchberger (Algoritmo 1 en la memoria), fundamentado en el Teorema 1.15, habitualmente llamado Criterio de Buchberger.

Presentamos en el capítulo 2, la conocida correspondencia entre Álgebra y Geometría, llamada en algunos textos “Diccionario entre el Álgebra y la Geometría", [8, 9]. Es también en este capítulo donde presentamos el concepto de ideales de dimensión 0. Diremos que un ideal es de dimensión 0 si el anillo cociente $k[x_1, \ldots, x_n]/I$ es un k-espacio vectorial de dimensión finita. Demostramos varios enunciados equivalentes a esta definición en el teorema 2.14. Incluye también este capítulo dos teoremas de Hilbert esenciales en el planteamiento algerogeométrico del estudio de soluciones de un sistema de ecuaciones algebraicas. Son el Teorema de la base (teorema 2.2) y el Teorema de los ceros, llamado también Nullstellensatz (teoremas 2.10 y 2.11).

Ilustrando los capítulos 1 y 2 se encuentran ejemplos realizados con SageMath (anteriormente SAGE). Incluimos en el Apéndice A un listado de comandos utilizados.

Los esfuerzos para mejorar la comlejidad del algoritmo de Buchberger no han tan éxitosos como se hubiera desea desde el punto de vista de las aplicaciones. Sin embargo, los ideales asociados a puntos (ideales de dimensión 0) han merecido la atención de matemáticos procedentes de distintas ramas ya que se usan en códigos, problemas de interpolación, biología molecular (véase [14] y sus referencias incluidas). La mejora para ideales de dimensión 0 se conoce con el nombre de Algoritmo de Möller-Buchberger (véase [15]). Su estudio detallado no se encuentra en esta memoria debido a la limitación del número de páginas requerido.

Nos hemos centrado en una solución posibilista ante el problema de complejidad del cálculo de una base de Göbner. Es el enfoque de los matemáticos Huang, England, Wilson, Bridge y Davenport publicado en 2019, [11]. En su trabajo se trata de resolver, con un enfoque posibilista, el problema del cómputo de una descomposición algebraica cilíndrica (CAD en sus siglas en inglés) de un conjunto algebraico real. La presentación de un tal conjunto involucra una familia finita de polinomios con coeficientes reales de la que se plantea, si es "rentable" calcular una base de Gröbner del ideal generado por esta familia. Este planteamiento, nos lleva a usar "Machine Learning" como aproximación al problema CAD para ideales de dimensión 0. Hemos trabajado el caso de dos variables, es decir puntos en $\mathbb{R}^2$ determinados implícitamente por una familia finita de polinomios bivariados.

Los conceptos básicos de "Machine Learning" se recogen en el capítulo 3, y un glosario de términos se halla en el apéndice B.

En el capítulo 4 se plantea el siguiente experimento: " Dada una familia de polinomios en dos variables con coeficientes computables decidir si es *rentable* o no calcular su base de Gröbner" . El criterio de retabilidad fijado está descrito en la Definición 4.1. Sin entrar en mucho detalle, diríamos que es rentable si en la base de Gröbner aparece un polinomio univariado. Los experimentos para entrenar han sido hechos añadiendo a SageMath algunos paquetes de "Machine Learning" que se encuentran detallados en el Apéndice C. Los resultados numéricos obtenidos son alentadores para el poco tiempo que hemos estado entrenando el modelo. Se encuentran recogigos en la sección 4.2, así como las conclusiones obtenidas de los mismos en la sección 4.3.

La memoria incluye una bibliografía, que no pretende ser exhaustiva, pero suficiente para justificar los objetivos fijados en este trabajo.

Todos los experimentos han sido realizados usando SageMath ([1]) con las ampliaciones de paquetes expuestas en el apéndice C.

CAPÍTULO 1
Órdenes monomiales

En este capítulo vamos a definir qué es un orden monomial y veremos su influencia a la hora de elegir una base de un ideal, dividir polinomios, etc. Hemos seguido las referencias [9, 8] para esta parte de la memoria. Definiremos el concepto de base de Gröbner de un ideal de un anillo de polinomios con coeficientes en cuerpo. Demostraremos el Algoritmo de Buchberger (Algoritmo 1), principal herramienta para calcular de forma efectiva una tal base. Se incluye un ejemplo para ilustrar estos conceptos (véase Ejemplo 1.19).

1.1. Conceptos básicos

El objetivo de esta sección es el establecimiento de los conceptos básicos relativos a la ordenación de los monomios en un anillo de polinomios. Demostraremos el Lema de Dickson (Teorema 1.6). Revisaremos los conceptos de SageMath relativos a los tipos de órdenes.

Definición 1.1. Un *orden monomial sobre* $k[x_1, ..., x_n]$ es una relación $>$ definida sobre el conjunto de los monomios x^α de $k[x_1, ..., x_n]$ que satisface:

1. $>$ es una relación de orden total (lineal).
2. $>$ es compatible con la multiplicación en $k[x_1, ..., x_n]$, en el sentido de que si $x^\alpha > x^\beta$ y x^γ es un monomio, entonces $x^\alpha x^\gamma = x^{\alpha+\gamma} > x^{\beta+\gamma} = x^\beta x^\gamma$.
3. $>$ es un buen order. Esto es, que toda colección de monomios no vacía tiene un elemento mínimo bajo la relación $>$.

A continuación presentamos los órdenes monomiales más utilizados en esta memoria.

Definición 1.2.
1. *Orden lexicográfico.* Sean x^α y x^β monomios sobre $k[x_1, ..., x_n]$. Decimos que $x^\alpha >_{lex} x^\beta$ si en la diferencia $\alpha - \beta \in \mathbb{Z}_n$, la entrada de más a la izquierda distinta de cero es positiva.
2. *Orden lexicográfico graduado.* Sean x^α y x^β monomios sobre $k[x_1, ..., x_n]$. Decimos que $x^\alpha >_{grlex} x^\beta$ si $\sum_{i=1}^n \alpha_i > \sum_{i=1}^n \beta_i$, o $\sum_{i=1}^n \alpha_i = \sum_{i=1}^n \beta_i$ y $x^\alpha >_{lex} x^\beta$.

3. *Orden lexicográfico inverso graduado.* Sean x^α y x^β monomios sobre $k[x_1, ..., x_n]$. Decimos que $x^\alpha >_{grevlex} x^\beta$ si $\sum_{i=1}^n \alpha_i > \sum_{i=1}^n \beta_i$, o $\sum_{i=1}^n \alpha_i = \sum_{i=1}^n \beta_i$ y en la diferencia $\alpha - \beta \in \mathbb{Z}_n$, la entrada de más a la derecha distinta de cero es negativa.

4. *Orden asociado a una forma lineal.* Todos los órdenes monomiales anteriores pueden ser expresado mediante matrices. Dada una matriz M por filas:

$$M = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix}.$$

   Podemos comparar dos monomios x^α y x^β sobre $k[x_1, ..., x_n]$, de la siguiente manera (empezando con $i = 1$):

   a) Comparamos sus pesos respecto a w_i ($\alpha \cdot w_i$ y $\beta \cdot w_i$). Si $\alpha \cdot w_i > \beta \cdot w_i$ o si $\beta \cdot w_i > \alpha \cdot w_i$, ordenamos los monomios de manera acorde.

   b) En caso de que se dé $\alpha \cdot w_i = \beta \cdot w_i$, si $i = n$ los dos monomios son iguales respecto al orden dado. Si $i < n$, volvemos al paso anterior, pero calculamos respecto a la siguiente fila. Es decir, volvemos al paso anterior tomando $i = i + 1$.

En SageMath, para definir un anillo de polinomios, utilizamos `PolynomialRing()` que es una función que tiene como argumentos de entrada obligatorios un anillo base y el número de variables. Pero también tiene algunos argumentos opcionales, que, entre otras cosas, nos permiten determinar el orden monomial que queremos utilizar. Para ello, tenemos que especificar `order`. Un ejemplo para generar un anillo de polinomios con coeficientes racionales (`QQ`), sobre dos variables x e y y con orden lexicográfico sería:

```
P.<x,y> = PolynomialRing(QQ, 2, order='lex')
```

El argumento `order`, puede ser:

- `'lex'`: orden lexicográfico.
- `'deglex'`: orden lexicográfico graduado.
- `'degrevlex'` (utilizado por defecto): orden lexicográfico inverso graduado.
- `'invlex'`: orden lexicográfico invertido.
- `'neglex'`: orden lexicográfico negativo.
- `'negdegrevlex'`: orden lexicográfico inverso graduado negativo.
- `'negdeglex'`: orden lexicográfico graduado negativo.

- Un objeto de la clase `TermOrder`: para definir órdenes más complejos. Podemos, por ejemplo, combinar los órdenes anteriores por bloques. También, podemos introducir un vector de pesos para cualquiera de los anteriores, por ejemplo, un lexicográfico graduado ponderado `order=TermOrder('wdeglex',(1,2,3))`. También podemos generar un `TermOrder` a partir de una matriz `M` y luego introducirlo como argumento de entrada. Es decir, `order=TermOrder(M)`. Como podemos ver, este argumento tiene una gran flexibilidad, lo que nos brinda una gran cantidad de opciones.

Para información adicional, véase [1].

Ejemplo 1.3. A continuación, vamos a ilustrar los tres órdenes anteriores con dos ejemplos hechos en SageMath utilizando `TermOrder()`.

Generamos una matriz `M` mediante `M = matrix(3,[2,3,1,0,1,4,3,2,1])`, donde 3 es el número de filas y la lista que aparece después son los elementos que se dividen automáticamente entre las filas. Una vez generada la matriz, generamos el anillo de polinomios: `P.<x,y,z>= PolynomialRing(QQ, 3, order=TermOrder(M))`. En este caso hemos elegido coeficientes racionales (`QQ`) y 3 variables: x, y, z.

A continuación, vemos cómo ordena el orden generado los siguientes polinomios (presentamos los comandos de SageMath):

$$x^3 * y * z^7 >= x * y^8 * z^2 \tag{1.1}$$

$$x^3 * y * z^3 > \quad x^3 * y * z^2 \tag{1.2}$$

(1.1) nos devuelve `False`, ya que al multiplicar los exponentes de ambos polinomios por la primera fila de `M` ($w_1 = [2, 3, 1]$ 1.2), el resultado es 16 y 28, respectivamente. Luego el segundo es mayor. En cuanto a (1.2) nos devuelve `True`. La misma operación que antes nos da, en este caso, 12 y 11, respectivamente. Luego el primero es mayor.

Añadir, que con la matriz `M` también podemos crear órdenes conocidos. Por ejemplo, utilizando la matriz: `M = matrix(3,[1,0,0,0,1,0,0,0,1])`, obtenemos el orden lexicográfico. Y, comparando los mismos polinomios:

$$x^3 * y * z^7 >= x * y^8 * z^2 \tag{1.3}$$

$$x^3 * y * z^3 > \quad x^3 * y * z^2 \tag{1.4}$$

En este caso (1.3) nos devuelve `True` debido a que $3 > 1$ (de los exponentes respectivos de las x), en concreto se cumple la desigualdad $>$. Y, (1.4) nos devuelve también `True` ya que los exponentes de las x y las y coinciden pero el de las z es 3 y 2, respectivamente. Luego, el primero es mayor.

A continuación nos centraremos en los ideales generados por monomios.

Definición 1.4. Un ideal $I \subseteq k[x_1, ..., x_n]$ es un ideal monomial si existe un subconjunto $A \subseteq \mathbb{Z}^n_{\geq 0}$ (posiblemente infinito) de manera que I está formado por todos los polinomios que son sumas finitas de la forma $\sum_{\alpha \in A} h_\alpha x^\alpha$, donde $h_\alpha \in k[x_1, ..., x_n]$. Y, lo denotamos $I = \langle x^\alpha | \alpha \in A \rangle$

Lema 1.5. *Sea $I = \langle x^\alpha | \alpha \in A \rangle$ un ideal monomial. Entonces, un monomio x^β pertenece a I si y solo si x^β es divisible por x^α para algún $\alpha \in A$.*

Demostración. Si x^β es múltiplo de x^α para algún $\alpha \in A$, entonces, por definición de ideal, $x^\beta \in I$.

Veamos ahora el recíproco. Si $x^\beta \in I$, entonces, podemos escribir $x^\beta = \sum_{i=1}^{s} h_i x^{\alpha(i)}$, con $h_i \in k[x_1, ..., x_n]$ y $\alpha(i) \in A$. Y, desarrollando cada h_i como suma de términos, obtenemos:

$$x^\beta = \sum_{i=1}^{s} h_i x^{\alpha(i)} = \sum_{i,j} c_{i,j} x^{\beta(i,j)} x^{\alpha(i)}$$

Finalmente, agrupando los términos del mismo multigrado, todo término de la parte derecha de la igualdad es divisible por algún $x^{\alpha(i)}$. Por lo tanto, el lado izquierdo de la igualdad ha de tener la misma propiedad. □

Teorema 1.6 (Lema de Dickson)**.** *Sea $I = \langle x^\alpha | \alpha \in A \rangle \subseteq k[x_1, ..., x_n]$ un ideal monomial. Entonces, I se puede escribir de la forma $I = \langle x^{\alpha(1)}, ..., x^{\alpha(s)} \rangle$, donde $\alpha(1), ..., \alpha(s) \in A$. En particular, I tiene una base monomial finita.*

Demostración. Por inducción sobre el número de variables, n.

Si $n = 1$, entonces I está generado por los monomios x_1^α, con $\alpha \in A \subseteq \mathbb{Z}_{\geq 0}$. Sea β el elemento más pequeño de $A \subseteq \mathbb{Z}_{\geq 0}$. Entonces, $\beta \leq \alpha$ para todo $\alpha \in A$. De esta manera, x_1^β divide a todos los otros generadores x_1^α. De lo cual se sigue que $I = \langle x_1^\beta \rangle$.

Asumamos ahora que $n > 1$ y que el teorema es cierto para $n-1$. Vamos a escribir las variables como $x_1, ..., x_{n-1}, y$, de manera que los monomios en $k[x_1, ..., x_{n-1}, y]$ se pueden escribir como $x^\alpha y^m$, donde $\alpha = (\alpha_1, ..., \alpha_{n-1}) \in \mathbb{Z}_{\geq 0}^{n-1}$ y $m \in \mathbb{Z}_{\geq 0}$.

Supongamos que $I \subseteq k[x_1, ..., x_{n-1}, y]$ es un ideal monomial. Para encontrar sus generadores, tomamos el ideal J en $k[x_1, ..., x_{n-1}]$ generado por los monomios x^α para los que $x^\alpha y^m \in I$ para algún $m \geq 0$. Dado que J es un ideal monomial en $k[x_1, ..., x_{n-1}]$, nuestra hipótesis de inducción implica que está generado por un conjunto finito de x^α's. Digamos, pues, que $J = \langle x^{\alpha(1)}, ..., x^{\alpha(s)} \rangle$.

Para cada i entre 1 y s, la definición de J nos dice que $x^{\alpha(i)} y^{m_i} \in I$ para algún $m_i \geq 0$. Sea m el mayor de los m_i, para cada l entre 0 y $m-1$, consideramos el ideal $J_l \subseteq k[x_1, ..., x_{n-1}]$ generado por los monomios x^β tales que $x^\beta y^l \in I$. Usando la hipótesis de inducción, J_l tiene un conjunto generador finito de monomios, es decir, $J_l = \langle x^{\alpha_l(1)}, ..., x^{\alpha_l(s_l)} \rangle$.

Afirmamos que I está generado por los monomios de la siguiente lista:

$$\begin{aligned} &de\ J : x^{\alpha(1)} y^m, ..., x^{\alpha(s)} y^m, \\ &de\ J_0 : x^{\alpha_0(1)}, ..., x^{\alpha_0(s_0)}, \\ &de\ J_1 : x^{\alpha_1(1)} y, ..., x^{\alpha_1(s_1)} y, \\ &\qquad \vdots \\ &de\ J_{m-1} : x^{\alpha_{m-1}(1)} y^{m-1}, ..., x^{\alpha_{m-1}(s_{m-1})} y^{m-1}. \end{aligned}$$

Nótese que todo monomio de I es divisible por uno de los monomios de esta lista. Para verlo, tomemos $x^\alpha y^p \in I$. Si $p \geq m$, entonces $x^\alpha y^p$ es divisible por algún $x^{\alpha(i)}y^m$ por la forma en la que hemos construido J. Por otro lado, si $p \leq m-1$, entonces $x^\alpha y^p$, es divisible por algún $x^{\alpha_p(j)}y^p$ por la forma en la que hemos construido J_p. Por 1.5, los monomios de arriba generan un ideal que tiene los mismos monomios que I. Por lo tanto, han de ser el mismo ideal. De esta manera, la afirmación ha quedado probada.

Para completar la prueba, solo falta demostrar que el conjunto de generadores finito puede ser elegido dado un conjunto de generadores del ideal. Si volvemos a escribir las variables como $x_1, ..., x_n$, entonces, nuestro ideal monomial es el siguiente: $I = \langle x^\alpha \mid \alpha \in A\rangle \subseteq k[x_1, ..., x_n]$. Tenemos que demostrar que I está generado por un conjunto finito de x^α's, donde $\alpha \in A$. Por el párrafo anterior, sabemos que $I = \langle x^{\beta(1)}, ..., x^{\beta(s)}\rangle$ para algunos monomios $x^{\beta(i)} \in I$. Dado que $x^{\beta(i)} \in I = \langle x^\alpha \mid \alpha \in A\rangle$, por 1.5, sabemos que cada $x^{\beta(i)}$ es divisible por $x^{\alpha(i)}$ para algún $\alpha(i) \in A$. Desde aquí, es fácil obtener que $I = \langle x^{\alpha(1)}, ..., x^{\alpha(s)}\rangle$. Lo cual completa la demostración. □

Finalmente, mencionaremos en esta sección el famoso Teorema de división de un polinomio f por un conjunto finito de polinomios $g_1, \ldots, g_t$. No incluimos su demostración por problemas de espacio. Se encuentra en [9], pag. 61.

Teorema 1.7. *Fijemos una anillo de polinomios $k[x_1, \ldots, x_n]$ y un orden monomial en él. Sean $(g_1, \ldots, g_t)$ una t-upla ordenada de polinomios. Entonces, para cada polinomio $f \in k[x_1, \ldots, x_n]$, existen $a_1, \ldots, a_t, r$ polinomios tales que*

$$f = a_1 g_1 + \cdots + a_t g_t + r \ , \tag{1.5}$$

con $r = 0$, o r combinación k-lineal de monomios x^α ninguno de los cuales es divisible por cualquiera de los monomios $LT(g_1), \ldots, LT(g_t)$. Además, si $a_i g_i \neq 0$, entonces $multideg(f) \geq multideg(a_i f_i)$.

Definición 1.8. Llamaranos al polinomio r obtenido según el teorema 1.7, *el resto de la división de f por $g_1, \ldots, g_t$.*

1.2. Bases de Gröbner

Definición 1.9. Sea $f = \sum_\alpha a_\alpha x^\alpha$ un polinomio no nulo sobre $k[x_1, ..., x_n]$ y $>$ un orden monomial, entonces definimos el multigrado de f como:

$$multideg(f) = \text{máx}(\alpha \in \mathbb{Z}^n_{\geq 0} \mid a_\alpha \neq 0)$$

Definición 1.10. Sea I un ideal $I \subseteq k[x_1, ..., x_n]$ distinto de $\{0\}$, y fijo un orden monomial sobre $k[x_1, ..., x_n]$. Entonces:

1. Denotamos por $LT(I)$ al conjunto de los términos principales de los elementos de I no nulos. Es decir,

$$LT(I) = \{cx^\alpha | \text{ tales que existe } f \in I \setminus \{0\} \text{ con } LT(f) = cx^\alpha\}$$

2. Denotamos por $\langle LT(I)\rangle$ al ideal generado por $LT(I)$.

Definición 1.11. Fijo un orden monomial sobre $k[x_1, ..., x_n]$. Un subconjunto finito $G = \{g_1, ..., g_t\}$ de un ideal $I \subseteq k[x_1, ..., x_n]$ distinto de $\{0\}$ se denomina *base de Gröbner* (o base estándar), si $\langle LT(g_1), ..., LT(g_t)\rangle = \langle LT(I)\rangle$.

Tomando que, por convención $\langle\emptyset\rangle = \{0\}$, definimos que la base de Gröbner del ideal $\{0\}$ como $\emptyset$.

Definición 1.12. Sean $f, g \in k[x_1, ..., x_n]$ no nulos. Fijo un orden monomial y sean $LT(f) = cx^\alpha$ y $LT(f) = dx^\beta$, con $c, d \in k$. Sea x^γ el $mcm(x^\alpha, x^\beta)$. El S-polinomio de f y g, que denotaremos por $S(f, g)$, es el polinomio:

$$S(f,g) = \frac{x^\gamma}{LT(f)}f - \frac{x^\gamma}{LT(g)}g$$

Una consecuencia de la definición 1.12 es que dados $G = \{g_1, ..., g_t\}$, si tenemos que $multideg(c_i x^{\alpha(i)} g_i) = multideg(c_j x^{\alpha(j)} g_j) = \delta$, entonces $S(x^{\alpha(i)} g_i, x^{\alpha(j)} g_j) = x^{\delta - \gamma_{ij}} S(g_i, g_j)$, donde $x^{\gamma_{ij}} = mcm(LM(g_i), LM(g_j))$.

Definición 1.13. Sea $G = \{g_1, ..., g_t\}$ una base de Gröbner, definimos $\overline{S(g_i, g_j)}^G$ como el resto de la división de $S(g_i, g_j)$ entre G.

Teorema 1.14 (Condición de la Cadena Ascendente)**.** *Sean*

$$I_1 \subseteq I_2 \subseteq I_3 \subseteq ...$$

una cadena ascendente de ideales en $k[x_1, ..., x_n]$. Entonces, existe un $N \geq 1$ tal que

$$I_N = I_{N+1} = I_{N+2} = ... \ .$$

Demostración. Dada una cadena ascendente de ideales $I_1 \subseteq I_2 \subseteq I_3 \subseteq ...$, consideramos el conjunto $I = \bigcup_{i=1}^{\infty} I_i$. Un cálculo fácil muestra que I es un ideal debido a que los ideales están encajados.

Fijamos un orden monomial, $<$. Entonces se tiene que, como la familia de ideales es una cadena, entonces $\langle LT(I)\rangle = \langle LT(I_i) | i \geq 1\rangle$. Por el Lema de Dickson, Teorema 1.6, tenemos que:

$$\langle LT(I)\rangle = \langle x^{\alpha(1)}, \ldots, x^{\alpha(s)}\rangle,$$

con $x^{\alpha(k)} = LT(g_k)$ para algún $g_k \in I_{i_k}$ $(i_k \geq 1)$. Sea $\ell = \text{máx}\{i_1, \ldots, i_s\}$. Entonces $g_1, \ldots, g_s \in I_l$.

Sea $n > \ell$. Definimos la familia

$$\mathcal{F} = \{LT(f) \mid f \in I_n \setminus I_\ell\} \ .$$

Supongamos que $\mathcal{F} \neq \emptyset$. Sea g el mínimo de $\mathcal{F}$ (que existe por la definición 1.1). Entonces $g = LT(f)$ para un $f \in I_n \setminus I_\ell$. En consecuencia,

$$LT(f) = a_1 x^{\alpha(1)} + \cdots + a_s x^{\alpha(s)},$$

para unos polinomios $a_i \in k[x_1, \ldots, x_n]$. Entonces

$$h := f - a_1 g_1 - \cdots - a_s g_s \in I_n$$

Existen dos casos posibles: o bien $h \in I_\ell$, o bien $h \in I_n \setminus I_\ell$. En el primer caso $f \in I_\ell$, lo que es una contradicción. En el segundo, $LT(h) \in \mathcal{F}$. Pero $LT(h) = LT(f - a_1 g_1 - \cdots - a_s g_s) < LT(f)$ lo cual contradice la minimilidad de $LT(f)$. En consecuencia, $\mathcal{F} = \emptyset$, luego $I_n = I_\ell$. Lo que concluye la demostración. □

Teorema 1.15 (Criterio de Buchberger)**.** *Un conjunto finito $G = \{g_1, ..., g_t\}$ es una base de Gröbner de $I = \langle g_1, ..., g_t \rangle$ si y solo si $\overline{S(g_i, g_j)}^G = 0$ para todo $i \neq j$.*

Demostración. $\Rightarrow$) Sea G una base de Gröbner, entonces, dado que $S(g_i, g_j) \in I$, tenemos que $S(g_i, g_j) = S(g_i, g_j) + 0$. Por lo tanto, $\overline{S(g_i, g_j)}^G = 0$.

$\Leftarrow$) Sea $f \in I$ no nulo, veamos que $LT(f) \in \langle LT(g_1), ..., LT(g_t) \rangle$. Podemos escribir $f = \sum_{i=1}^t h_i g_i$ con $h_i \in k[x_1, ..., x_n]$. Y, $multideg(f) \leq max(multideg(h_i g_i) | h_i g_i \neq 0)$.

Tomamos la representación de f tal que $\delta = max(multideg(h_i g_i) | h_i g_i \neq 0)$ sea mínima (la cual sabemos que existe porque nuestro orden monomial es un buen orden). Por lo tanto, obtenemos que $multideg(f) \leq \delta$.

Si se da la igualdad, $LT(f)$ es divisible entre algún $LT(g_i)$, y tendríamos que $LT(f) \in \langle LT(g_1), ..., LT(g_t) \rangle$ (que es lo que queremos probar).

Por lo tanto, nos falta ver el caso en que $multideg(f) < \delta$. Sea $f = \sum_{i=1}^t h_i g_i$ con δ mínimo, podemos escribir f como sigue:

$$\begin{aligned} f &= \sum_{multideg(h_i g_i) = \delta} h_i g_i + \sum_{multideg(h_i g_i) < \delta} h_i g_i \\ &= \sum_{multideg(h_i g_i) = \delta} LT(h_i) g_i + \sum_{multideg(h_i g_i) = \delta} (h_i - LT(h_i)) g_i + \sum_{multideg(h_i g_i) < \delta} h_i g_i \end{aligned} \tag{1.6}$$

En esta suma, los términos segundo y tercero de la segunda línea tienen $multideg < \delta$. Entonces, $multideg(f) < \delta$ significa que el primer término de la segunda línea también tiene que cumplir $multideg < \delta$.

Por otro lado, tenemos que $S(LT(h_i)g_i, LT(h_j)g_j) = x^{\delta - \gamma_{ij}} S(g_i, g_j)$ (por la definición 1.12).

De donde se sigue que el término $\sum_{multideg(h_i g_i)} LT(h_i) g_i$ es una combinación lineal de elementos de la forma $x^{\delta - \gamma_{ij}} S(g_i, g_j)$ para ciertos pares (i, j). Ahora, considerando uno de estos $S(g_i, g_j)$ (dado que $\overline{S(g_i, g_j)}^G = 0$) el algoritmo de la división nos permite escribir $S(g_i, g_j) = \sum_{l=1}^t A_l g_l$, con $A_l \in k[x_1, ..., x_n]$ y $multideg(A_l g_l) \leq multideg(S(g_i, g_j))$ cuando $A_l g_l \neq 0$. Ahora, multiplicamos cada lado, por $x^{\delta - \gamma_{ij}}$ y obtenemos $x^{\delta - \gamma_{ij}} S(g_i, g_j) = \sum_{l=1}^t B_l g_l$, con $B_l = x^{\delta - \gamma_{ij}} A_l$. Cuando $B_l g_l \neq 0$ tenemos

$$multideg(B_l g_l) < multideg(x^{\delta - \gamma_{ij}} S(g_i, g_j)) < \delta \tag{1.7}$$

ya que $LT(S(g_i, g_j)) < mcm(LM(g_i), LM(g_j))$.

De donde se sigue que el primer término de la segunda línea de (1.6) es una combinación lineal de elementos $x^{\delta-\gamma_{ij}}S(g_i, g_j)$ que satisfacen las propiedades que acabamos de ver. Luego, podemos escribir $\sum_{multideg(h_i g_i)=\delta} LT(h_i)g_i = \sum_{l=1}^{t} \tilde{B}_l g_l$, y cuando $multideg(\tilde{B}_l g_l) \neq 0$ tenemos $multideg(\tilde{B}_l g_l) < \delta$.

Finalmente, sustituyendo la expresión anterior en (1.6), obtenemos una expresión para f como combinación de los polinomios g_i, donde todos los términos tiene $multideg < \delta$. Lo cual contradice la minimalidad de δ, completando así la prueba.

□

Teorema 1.16 (Algoritmo de Buchberger)**.** *Sea $I = \langle f_1, ..., f_s\rangle \neq \{0\}$ un ideal polinomial. Una base de Gröbner de I puede ser construida en un número finito de pasos, siguiendo el siguiente algoritmo:*

Algoritmo 1: Algoritmo de Buchberger

Input : $F = (f_1, ..., f_s)$
Output: *base de Gröbner* $G = g_1, ..., g_s$ *de* $I = \langle F\rangle$, *con* $F \subseteq G$

Initialize $G := F$;
Initialize $G' := \emptyset$;
while $G \neq G'$ **do**
  $G' := G$;
  foreach *pair* $p \neq q$ **in** G' **do**
    $S := \overline{S(p,q)}^{G'}$ *1.12*;
    if $S \neq 0$ **then**
      $G := G \cup \{S\}$;
    end
  end
end
return G

La base de Gröbner buscada es G.

Demostración. Vamos a emplear la siguiente notación. Si $G = \{g_1, ..., g_t\}$, entonces $\langle G\rangle$ y $\langle LT(G)\rangle$ denotan los siguiente ideales:

$$\langle G\rangle = \langle g_1, ..., g_t\rangle$$
$$\langle LT(G)\rangle = \langle LT(g_1), ..., LT(g_t)\rangle$$

Volviendo a la demostración del teorema, primero vamos a demostrar que $G \subseteq I$ se cumple en todas las etapas del algoritmo. Esto es cierto inicialmente, y, cuando aumentamos G, los hacemos añadiendo el resto $r = \overline{S(p,q)}^{G'}$ para $p, q \in G' \subseteq G$. Por lo tanto, si $G' \subseteq I$, entonces p, q y, por lo tanto, $S(p, q)$ están en I. Y dado que estamos dividiendo por $G' \subseteq I$, obtenemos $G \cup \{r\} \subseteq I$. Nótese también que G contiene la base F de I, y, por consiguiente, G es una base de I.

El algoritmo termina cuando $G = G'$, lo que significa que $r = \overline{S(p,q)}^{G'} = 0$ para todo $p, q \in G$. Por lo tanto, G es una base de Gröbner de $\langle G \rangle = I$ por 1.15.

Solo falta demostrar que el algoritmo termina. Necesitamos considerar qué pasa después de cada paso del bucle principal. El conjunto G está formado por G' (el G anterior) unión los restos distintos de cero de los S-polinomios de elementos de G'. Entonces

$$\langle LT(G') \rangle \subseteq \langle LT(G) \rangle \tag{1.8}$$

ya que $G' \subseteq G$. De hecho, si $G' \neq G$, afirmamos que l contenido de (1.8) es estricto, es decir, no se puede dar la igualdad. Para ver esto, supongamos que un resto distinto de cero, r, de un S-polinomio ha sido añadido a G. Dado que r es un resto de división por G', $LT(r)$ no es divisible por ningún término principal de elementos de G', y por lo tanto, $LT(r) \notin \langle LT(G') \rangle$ por 1.5. Aunque, de hecho, $LT(r) \in \langle LT(G) \rangle$, lo cual demuestra la afirmación.

Por (1.8), los ideales $\langle LT(G') \rangle$ de sucesivas iteraciones del bucle forman una cadena ascendente de ideales en $k[x_1, ..., x_n]$. Por lo tanto, el 1.14 implica que, después de un número finito de iteraciones, la cadena se estabilizará, de tal manera que $\langle LT(G') \rangle = \langle LT(G) \rangle$ tendrá que darse eventualmente. Por el párrafo anterior, esto implica que $G' = G$, y por lo tanto, el algoritmo debe terminar tras un número finito de pasos. □

Definición 1.17. Dados polinomios no nulos $F = (f_1, ..., f_s)$, decimos que

$$S(f_i, f_j) = \sum_{l=1}^{s} A_l f_l$$

es una *representación mcm* cuando cumple que

$$mcm(LM(f_i), LM(f_j)) > LT(A_l f_l) \text{ siempre que } A_l f_l \neq 0.$$

Teorema 1.18. *Una base $G = g_1, ..., g_n$ de un ideal I es una base de Gröbner si y solo si para todo $i \neq j$, el S-polinomio $S(g_i, g_j)$ tiene una representación mcm.*

Demostración. Si G es una base de Gröbner, entonces todo S-polinomio tiene una representación estándar, y por consiguiente una representación mcm. Para ver el recíproco, vamos a seguir la prueba de 1.15.

Estamos asumiendo que $S(g_i, g_j)$ tiene una representación mcm

$$S(g_i, g_j) = \sum_{l=1}^{t} A_l g_l$$

donde $x^{\gamma_{ij}} > LT(A_l g_l)$ cuando $A_l g_l \neq 0$. Con $x^{\gamma_{ij}} = mcm(LM(g_i), LM(g_j))$. Si definimos $B_l = x^{\delta - \gamma_{ij}} A_l$, entonces

$$x^{\delta - \gamma_{ij}} S(g_i, g_j) = \sum_{l=1}^{t} B_l g_l,$$

donde

$$multideg(B_l g_l) = multideg(x^{\delta-\gamma_{ij}}) + multideg(A_l g_l) < (\delta - \gamma_{ij}) + \gamma_{ij} = \delta$$

Lo cual nos lleva a la misma desigualdad (1.7) a la que llegábamos en la prueba 1.15. Desde aquí, el resto de la demostración es idéntico a lo que se hizo en 1.15, completando la prueba del teorema. □

Ejemplo 1.19. A continuación, vamos a ver un ejemplo en SageMath de cálculo de una base de Gröbner de un ideal dado.

Primero, hay que generar el anillo de polinomios. Por ejemplo,

```
P.<x,y>= PolynomialRing(QQ, 2, order='deglex').
```

Una vez generado el anillo de polinomios, creamos un ideal. Por ejemplo a partir de sus generadores:

```
I = ideal(5*x + 3*y - 1, x^2 + y^2- 1).
```

Ahora que tenemos el ideal I, podemos calcular una base de Gröbner utilizando

```
G = I.groebner_basis(),
```

que en este caso nos devuelve `G = [y^2 - 3/17*y - 12/17, x + 3/5*y - 1/5]`. Es decir, dado $I = \langle 5x + 3y - 1, x^2 + y^2 - 1\rangle \subseteq \mathbb{Q}[x, y]$ y fijo el orden lexicográfico graduado. Una base de Gröbner, G, del ideal, I, con el orden que hemos fijado es $G = \left\{y^2 - \frac{3}{17}y - \frac{12}{17}, x + \frac{3}{5}y - \frac{1}{5}\right\}$.

Para cambiar el orden respecto al cual queremos calcular la base de Gröbner, tenemos que generar un nuevo anillo de polinomios con `PolynomialRing()` especificando el orden respecto al cual queremos calcular la base con el parámetro `order` y repetir los pasos anteriores. Los valores para este parámetro son muy flexibles y veíamos algunos en 1.2.

CAPÍTULO 2
Ideales de polinomios

En este capítulo presentamos los resultados básicos sobre anillos de polinomios que necesitaremos para dar una formalización básica a la interpretación geométrica de los conjuntos de ceros de ideales generados por una familia finita o no de polinomios. Consideraremos en este capítulo que los coeficientes de los mismos están en un cuerpo k. No es necesario a priori que k sea algebraícamente cerrado, ni tampoco de característica 0. Señalaremos en cada caso cuando estas hipótesis sean necesarias.

2.1. El Teorema de la base de Hilbert

En esta sección k denotará un cuerpo, y $x_1, \ldots, x_n$ variables sobre k. Denoteremos por $k[x_1, \ldots, x_n]$ al anillo de polinomios en las variables $x_1, \ldots, x_n$ con coeficientes en k.

Proposición 2.1. Sea I un ideal no nulo de $k[x_1, \ldots, x_n]$.

1. $\langle LT(I)\rangle$ es un ideal monomial.

2. Existen $g_1, ..., g_t \in I$ tales que $\langle LT(I)\rangle = \langle LT(g_1), ..., LT(g_t)\rangle$.

Demostración. a) Los monomios principales $LM(g)$ de elementos $g \in I \setminus \{0\}$ generan el ideal monomial $\langle LM(g) \mid g \in I \setminus \{0\}\rangle$. Dado que $LM(g)$ y $LT(g)$ difieren por una constante distinta de cero, este ideal es igual a $\langle LT(g) \mid g \in I \setminus \{0\}\rangle = \langle LT(I)\rangle$. Luego, $LT(I)$ es un ideal monomial.

b) Dado que $\langle LT(I)\rangle$ está generado por los monomios $LM(g)$ para $g \in I \setminus \{0\}$, el Lema de Dickson 1.6 nos dice que $\langle LT(I)\rangle = \langle LM(g_1), ..., LM(g_t)\rangle$ para un número finito de $g_1, ..., g_t \in I$. Dado que $LM(g_i)$ y $LT(g_i)$ difieren por una constante distinta de cero, se sigue que $\langle LT(I)\rangle = \langle LT(g_1), ..., LT(g_t)\rangle$. Completando así la demostración. □

Teorema 2.2 (Teorema de la base de Hilbert)**.** *Todo ideal $I \subseteq k[x_1, ..., x_n]$ tiene un conjunto generador finito. Es decir, $I = \langle g_1, ..., g_t\rangle$ con $g_1, ..., g_t \in I$.*

Demostración del teorema de la base de Hilbert. Si $I = (0)$, tomamos como conjunto generador $\{0\}$, que es finito. Ahora, sea I un ideal que contiene algún polinomio no nulo, el conjunto generador $g_1, ..., g_s$ de I se puede construir de la siguiente manera:

Primero, seleccionamos un orden monomial para utilizarlo en nuestro algoritmo de la división y en el cómputo de términos principales (LT). Entonces, I tiene un ideal de términos principales $\langle LT(I)\rangle$. Por 2.1, existen $g_1, ..., g_t \in I$ tales que $\langle LT(I)\rangle = \langle LT(g_1), ..., LT(g_t)\rangle$. Afirmamos que $I = \langle g_1, ..., g_t\rangle$.

Es evidente que $\langle g_1, ..., g_t\rangle \subseteq I$ ya que cada $g_i \in I$. Recíprocamente, sea $f \in I$ un polinomio cualquiera. Si aplicamos el algoritmo de la división para dividir f por $(g_1, ..., g_t)$, obtenemos una expresión como la siguiente:

$$f = q_1 g_1 + ... + q_t g_t + r$$

donde ningún término de r es divisible por ningún $LT(g_1), ..., LT(g_t)$. Afirmamos que $r = 0$.

Para comprobarlo, veamos que:

$$r = f - q_1 g_1 - ... - q_t g_t \in I.$$

Si $r \neq 0$, entonces $LT(r) \in \langle LT(I)\rangle = \langle LT(g_1), ..., LT(g_t)\rangle$, y, por 1.5, $LT(r)$ tendría que ser divisible por algún $LT(g_i)$. Lo cual contradice el concepto de resto. Por lo tanto, r debe ser 0. Entonces:

$$f = q_1 g_1 + ... + q_t g_t + 0 \in \langle g_1, ..., g_t\rangle,$$

lo cual prueba que $I \subseteq \langle g_1, ..., g_t\rangle$, completando la prueba. □

A continuación, vamos a ver un ejemplo para ilustrar que la condición de que el número de variables sea finito, es decir, $I \subseteq k[x_1, ..., x_n]$ de 2.2 es una condición necesaria para que se cumpla el teorema.

Ejemplo 2.3. En este ejemplo vamos a tomar un anillo de polinomios sobre infinitas variables, y vamos a demostrar que no existe un conjunto generador finito para cierto ideal.

Sea k un cuerpo y $\{x_j\}_{j=1}^{\infty}$ un conjunto numerable de variables sobre k. Definimos el conjunto R como

$$f \in R \iff \exists N \in \mathbb{N} \text{ con } f \in k[x_1, \ldots, x_N] \ .$$

El conjunto R así definido tiene estructura de anillo, y podríamos decir que es *el anillo de polinomios en un conjunto numerable de variables con coeficientes en* k. Lo denotaremos por $R = k[x_1, x_2, \ldots]$.

Sea $I \subseteq k[x_1, x_2, ...]$, consideremos el ideal engendrado por todas las variables $I = \langle x_1, x_2, ...\rangle$, es decir

$$I = \{f \in R \mid \exists N \text{ con } f = a_1 x_1 + \cdots + a_N x_N \ , \ a_\ell \in k[x_1, \ldots, x_N]\} \tag{2.1}$$

Es fácil demostrar que I es un ideal, veamos que I no es de generación finita. Para esto, supongamos que tiene un conjunto generador finito, digamos $I = \langle g_1, ..., g_t \rangle$ con $g_1, ..., g_t \in I$. Dado que los polinomios $g_1, ..., g_t$ solo pueden depender de un número finito de variables, existe un N tal que en $k[x_1, \ldots, x_N]$ se tienen las igualdades

$$g_i = f_{i1}\, x_1 + f_{i2}\, x_2 + \cdots + f_{iN}\, x_N \ , \ i = 1, \ldots, t. \tag{2.2}$$

Sea M un número natural mayor estrictamente que N. Se tiene que x_M está en I, luego

$$x_M = a_1 g_1 + \cdots + a_t g_t \ \text{ con } a_i \in k[x_1, x_2 \ldots] \ . \tag{2.3}$$

Sea m un número natural tal que $a_i \in k[x_1, \ldots, x_m]$ para todo i. Sea $r = \text{máx}\{M, m\}$. Entonces, la igualdad (2.3) se da en el anillo de polinomios $k[x_1, \ldots, x_r]$. Definimos el homomorfismo de anillos $\phi : k[x_1, \ldots, x_r] \to k$ de la siguiente manera:

$$\phi(x_M) = 1, \ \phi(x_i) = 0 \text{ si } i \neq M, \ \phi(\lambda) = \lambda, \ \forall \lambda \in k \ .$$

Aplicando ϕ a la igualdad (2.3) obtenemos:

$$1 = \phi(x_M) = \phi(a_1)\phi(g_1) + \cdots + \phi(a_t)\phi(g_t) = 0 \ .$$

Luego hemos llegado a una contradicción, y, por lo tanto, I no es un ideal finitamente generado.

2.2. Diccionario entre el Álgebra y la Geometría

En esta sección presentaremos la correspondencia que existe entre ideales de un anillo de polinomios en n variables y el conjunto de soluciones en k^n de un sistema de ecuaciones polinómicas.

Definición 2.4. Sea I un ideal de $k[x_1, ..., x_n]$. El radical de I es el conjunto:

$$\sqrt{I} = \{g \in k[x_1, ..., x_n] \mid g^m \in I \ \textit{para algún}\ m \geq 1\}$$

Se dice que I es un ideal radical si $\sqrt{I} = I$.

Es fácil demostrar que $\sqrt{I}$ es un ideal.

Definición 2.5. Sean $f_1, \ldots, f_s \in k[x_1, \ldots, x_n]$. El conjunto de soluciones simultáneas $(a_1, ..., a_n) \in k^n$ a un sistema de ecuaciones

$$f_1(x_1, ..., x_n) = 0, \ldots, f_s(x_1, ..., x_n) = 0 \ ,$$

es denominado *variedad afín* definida por $f_1, ..., f_s$, y se denota por $V(f_1, ..., f_s)$. Un subconjunto $V \subset k^n$ es denominado *variedad afín* si $V = V(f_1, ..., f_s)$ para algún conjunto de polinomios $f_i \in k[x_1, ..., x_n]$.

Definición 2.6. Sea $I \subseteq k[x_1, ..., x_n]$ un ideal. Denotaremos por $V(I)$ al conjunto

$$V(I) = \{(a_1, ..., a_n) \in k^n | f(a_1, ..., a_n) = 0 \text{ para todo } f \in I\}$$

y lo llamaremos *variedad afín definida por el ideal* I.

Proposición 2.7. Se tiene que $V(I)$ es una variedad afín. En particular, si $I = \langle f_1, ..., f_s \rangle$, entonces $V(I) = V(f_1, ..., f_s)$.

Demostración. Por 2.2, $I = \langle f_1, ..., f_s \rangle$ para un conjunto de generadores finito. Afirmamos que $V(I) = V(f_1, ..., f_s)$. Primero, dado que $f_i \in I$, si $f(a_1, ..., a_n) = 0$ para todo $f \in I$, entonces $f_i(a_1, ..., a_n) = 0$, y por consiguiente $V(I) \subseteq V(f_1, ..., f_s)$. Por otro lado, sea $(a_1, ..., a_n) \in V(f_1, ..., f_s)$ y sea $f \in I$. Dado que $I = \langle f_1, ..., f_s \rangle$, podemos escribir

$$f = \textstyle\sum_{i=1}^{s} h_i f_i$$

para algunos $h_i \in k[x_1, ..., x_n]$. Pero, entonces

$$f(a_1, ..., a_n) = \textstyle\sum_{i=1}^{s} h_i(a_1, ..., a_n) f_i(a_1, ..., a_n) = \sum_{i=1}^{s} h_i(a_1, ..., a_n) \cdot 0 = 0$$

Y, por lo tanto, $V(f_1, ..., f_s) \subseteq V(I)$. Quedando demostrada la proposición. □

Definición 2.8. Sea $V \subset k^n$ una variedad afín. Denotamos por $I(V)$ el conjunto:

$$I(V) = \{f \in k[x_1, ..., x_n] \mid f(a_1, ..., a_n) = 0 \text{ para todo } (a_1, ..., a_n) \in V\}$$

Este conjunto es denominado *ideal de* V.

Notación. Sea $f \in k[x_1, \ldots, x_n]$ y $a \in k$. Denotaremos por $\overline{f}(x_1, \ldots, x_{n-1}) = f(x_1, \ldots, x_{n-1}, a)$. También, si I es un ideal de $k[x_1, \ldots, x_n]$, definimos

$$I_{x_n=a} = \left\{\overline{f} \mid f \in I\right\} \tag{2.4}$$

Además se tiene que $I_{x_n=a}$ es un ideal de $k[x_1, \ldots, x_{n-1}]$. Análogamente podemos definir los ideales de evaluación:

$$I_{x_n=a_n,\ldots,x_{j+1}=a_{j+1}} \subset k[x_1, \ldots x_j] \tag{2.5}$$

donde $a_n, \ldots, a_{j+1} \in k$.

Con las notaciones anteriores podemos probar el siguiente resultado.

Proposición 2.9. Si k es un cuerpo algebraícamente cerrado e $I \subsetneq k[x_1, ..., x_n]$ es un ideal propio, entonces existe un $a \in k$ tal que $I_{x_n=a} \subsetneq k[x_1, ..., x_{n-1}]$.

Demostración. Para probar la proposición, dividimos la prueba en dos casos dependiendo del tamaño de $I \cap k[x_n]$.

Caso 1. $I \cap k[x_n] \neq \{0\}$. Sea $f \in I \cap k[x_n]$ distinto de cero y no constante (ya que si lo fuera, entonces, dado que $1 \in I \cap k[x_n] \subseteq I$, contradeciría $I \neq k[x_1, ..., x_n]$).

Dado que k es algrebraícamente cerrado

$$f = c\prod_{i=1}^{r}(x_n - b_i)^{m_i}, \text{ con } c, b_1, ..., b_r \in k \text{ y } c \neq 0.$$

Supongamos que $I_{x_n=b_i} = k[x_1, ..., x_{n-1}]$ para todo i. Entonces, para todo i, existe un $B_i \in I$ con $B_i(x_1, ..., x_{n-1}, b_i) = 1$. Lo cual implica que

$$1 = \prod_{i=1}^{r}(A_i(x_n - b_i) + B_i)^{m_i} = A\prod_{i=1}^{r}(x_n - b_i)^{m_i} + B,$$

donde $A = \prod_{i=1}^{r} A_i^{m_i}$ y $B \in I$. Esto y $\prod_{i=1}^{r}(x_n - b_i)^{m_i} = c^{-1}f \in I$ implica que $1 \in I$, lo cual contradice que $I \neq k[x_1, ..., x_n]$. Y, por lo tanto, $I_{x_n=b_i} \neq k[x_1, ..., x_{n-1}]$ para algún i. Ese b_i es el a buscado.

Caso 2. $I \cap k[x_n] = \{0\}$. Tomamos $G = \{g_1, ..., g_t\}$ una base de Gröbner de I respecto al orden lexicográfico con $x_1 > ... > x_n$ y escribimos:

$$g_i = c_i(x_n)x^{\alpha_i} + \textit{términos} < x^{\alpha_i}, \tag{2.6}$$

donde $c_i(x_n) \in k[x_n]$ es distinto de cero y x^{α_i} es un monomio en $x_1, ..., x_{n-1}$.

Ahora, elegimos un $a \in k$ tal que $c_i(a) \neq 0$ para todo i. Lo cual es posible porque los cuerpos algebraícamente cerrados son infinitos. Es fácil comprobar que los polinomios

$$\bar{g}_i = g_i(x_1, ..., x_{n-1}, a)$$

forman una base del ideal $I_{x_n=a}$. Sustituyendo $x_n = a$ en la ecuación (2.6), podemos ver fácilmente que $LT(\bar{g}_i) = c_i(a)x^{\alpha_i}$ ya que $c_i(a) \neq 0$. Además $x^{\alpha_i} \neq 1$, ya que, en caso contrario $g_i = c_i \in I \cap k[x_n] = \{0\}$, y contradeciría que $c_i \neq 0$. Esto demuestra que $LT(\bar{g}_i)$ no es constante para todo i.

Afirmamos que los $\bar{g}_i$ forman una base de Gröbner de $I_{x_n=a}$. Asumiendo esta afirmación, se sigue que $1 \notin I_{x_n=a}$, ya que ningún $LT(\bar{g}_i)$ puede dividir a 1. Por consiguiente, $I_{x_n=a} \neq k[x_1, ..., x_{n-1}]$ que es lo que queríamos demostrar.

Para probar esta última afirmación, tomamos $g_i, g_j \in G$ y consideramos el polinomio

$$S = c_j(x_n)\frac{x^\gamma}{x^{\alpha_i}}g_i - c_i(x_n)\frac{x^\gamma}{x^{\alpha_j}}g_j$$

donde $x^\gamma = mcm(x^{\alpha_i}, x^{\alpha_j})$. Por construcción, $x^\gamma > LT(S)$. Como $S \in I$, tiene una representación estándar $S = \sum_{l=1}^{t} A_l g_l$. Entonces, evaluando en $x_n = a$, nos da

$$c_j(x_n)\frac{x^\gamma}{x^{\alpha_i}}\bar{g}_i - c_i(x_n)\frac{x^\gamma}{x^{\alpha_j}}\bar{g}_j = \bar{S} = \sum_{l=1}^{t}\bar{A}_l\bar{g}_l$$

Como $LT(\bar{g}_i) = c_i(a)x^{\alpha_i}$, vemos que $\bar{S}$ es el S-polinomio (1.12) $S(\bar{g}_i, \bar{g}_j)$, salvo la constante no nula $c_i(a)c_j(a)$. Entonces,

$$x^\gamma > LT(S) \geq LT(A_l g_l), \text{ con } A_l g_l \neq 0,$$

lo cual implica que

$$x^\gamma > LT(\bar{A}_l\bar{g}_l), \text{ con } \bar{A}_l\bar{g}_l \neq 0.$$

Dado que $x^\gamma = mcm(LM(\bar{g}_i, \bar{g}_j))$, se sigue que $S(\bar{g}_i, \bar{g}_j)$ tiene una representación mcm para todo i, j y, por 1.18, es una base de Gröbner. Esto prueba la afirmación, completando la prueba de la proposición. □

Teorema 2.10 (Teorema de los ceros de Hilbert "débil"). *Sea k un cuerpo algebraícamente cerrado y sea $I \subseteq k[x_1, ..., x_n]$ un ideal que satisface $V(I) = \emptyset$. Entonces,*

$$I = k[x_1, ..., x_n].$$

Demostración. Vamos a probar el contrarrecíproco, es decir:

$$I \subsetneq k[x_1, ..., x_n] \Rightarrow V(I) \neq \emptyset.$$

Para ello, será utilizada la siguiente equivalencia.

$$I = k[x_1, ..., x_n] \Longleftrightarrow 1 \in I \tag{2.7}$$

Lo cual es inmediato, ya que, por un lado, si $I = k[x_1, ..., x_n]$, entonces $1 \in I$. Y, por el otro lado, tenemos que $1 \in I$. Tomamos cualquier $r \in k[x_1, ..., x_n]$, y, dado que $r = r \cdot 1 \in I$, tenemos $k[x_1, ..., x_n] \subseteq I \subseteq k[x_1, ..., x_n]$. Luego, concluimos que $I = k[x_1, ..., x_n]$.

Dado $a \in k$ y $f \in k[x_1, ..., x_n]$, sea $\bar{f} = f(x_1, ..., x_{n-1}, a) \in k[x_1, ..., x_{n-1}]$, el conjunto:

$$I_{x_n=a} = \{\bar{f} \mid f \in I\} \subset k[x_1, ..., x_{n-1}]$$

es un ideal de $k[x_1, ..., x_{n-1}]$. La clave de la prueba está en la proposición 2.9. Mediante inducción, obtenemos $a_1, ..., a_n \in k$ tales que $I_{x_n=a_n,...,x_1=a_1} \subsetneq k$. Pero los únicos ideales de k son $\{0\}$ y k, luego $I_{x_n=a_n,...,x_1=a_1} = \{0\}$. Y, por consiguiente, $(a_1, ..., a_n) \in V(I)$. De lo cual, concluimos que $V(I) \neq \emptyset$, y queda demostrado el teorema 2.10. □

Teorema 2.11 (Teorema de los ceros de Hilber, Nullstellensatzt). *Sea k un cuerpo algebraícamente cerrado. Si $f, f_1, ..., f_s \in k[x_1, ..., x_n]$, entonces, $f \in I(V(f_1, ..., f_s))$ si y solo si*

$$f^m \in \langle f_1, ..., f_s \rangle$$

para algún entero $m \geq 1$. Equivalentemente, sea k un cuerpo algebraícamente cerrado, y sea I un ideal en $k[x_1, ..., x_n]$, entonces:

$$I(V(I)) = \sqrt{I}$$

Demostración. Dado un polinomio no nulo f que se anula en todo 0 común a los polinomios $f_1, ..., f_s$, debemos probar que existe un entero $m \geq 1$ y polinomios $A_1, ..., A_s$ tales que

$$f^m = \textstyle\sum_{i=1}^{s} A_i f_i.$$

Para ello, vamos a utilizar el siguiente truco. Consideremos el ideal:

$$\tilde{I} = \langle f_1, ..., f_s, 1 - yf \rangle \subseteq k[x_1, ..., x_n, y],$$

donde $f, f_1, ..., f_s$ son como las del teorema. Afirmamos que

$$V(\tilde{I}) = \emptyset.$$

Para comprobarlo, tomamos $(a_1, ..., a_n, a_{n+1}) \in k^{n+1}$. Se tienen que dar uno de los dos casos siguientes:

1. $(a_1, ..., a_n)$ es un 0 común a $f_1, ..., f_s$.
2. $(a_1, ..., a_n)$ no es un 0 común a $f_1, ..., f_s$.

En el primer caso $f(a_1, ..., a_n) = 0$ dado que f se anula en todo 0 común a $f_1, ..., f_s$. Por lo tanto, el polinomio $1 - yf$ toma el valor $1 - a_{n+1}f(a_1, ..., a_n) = 1 - 0 = 1 \neq 0$ en el punto $(a_1, ..., a_n, a_{n+1})$. En particular, $(a_1, ..., a_n, a_{n+1}) \notin V(\tilde{I})$. En el segundo caso, para algún i, $1 \leq i \leq s$, debemos tener $f_i(a_1, ..., a_n) = 0$. Pensando en f_i como una función de $n + 1$ variables (que no depende de la última variable), tenemos $f_i(a_1, ..., a_n, a_{n+1}) \neq 0$. En particular, volvemos a concluir que $(a_1, ..., a_n, a_{n+1}) \notin V(\tilde{I})$. Dado que $(a_1, ..., a_n, a_{n+1}) \in k^{n+1}$ es arbitrario, obtenemos que $V(\tilde{I}) = \emptyset$, tal y como habíamos afirmado.

Ahora, aplicamos teorema 2.10 para concluir que $1 \in \tilde{I}$. Por consiguiente

$$1 = \sum_{i=1}^{s} p_i(x_1, ..., x_n, y) f_i + q(x_1, ..., x_n, y)(1 - yf) \tag{2.8}$$

para algunos polinomios $p_i, q \in k[x_1, ..., x_n, y]$. Ahora, fijamos $y = 1/f(x_1, ..., x_n)$. Entonces, (2.8) implica que

$$1 = \textstyle\sum_{i=1}^{s} p_i(x_1, ..., x_n, 1/f) f_i.$$

Multiplicamos ambos lados por una potencia f^m, donde m es elegido de manera que es lo suficientemente grande para eliminar los denominadores. Esto resulta en

$$f^m = \textstyle\sum_{i=1}^{s} A_i f_i,$$

para algunos polinomios $A_i \in k[x_1, ..., x_n]$, que es lo que queríamos probar. □

2.3. Ideales de dimensión 0

El objetivo de esta sección es la caracterización de los ideales del anillo $k[x_1, \ldots, x_n]$ que corresponden a un conjunto finito de puntos del espacio afín k^n. Es el llamado Teorema de finitud, Teorema 2.14 de esta memoria.

Proposición 2.12. Fijo un orden monomial en $k[x_1, ..., x_n]$ y un ideal $I \subseteq k[x_1, ..., x_n]$, se tiene que:

1. Todo $f \in k[x_1, ..., x_n]$ es congruente módulo I con un único polinomio r que es combinación k-lineal de monomios del complementario de $\langle LT(I)\rangle$.

2. Los elementos del conjunto $\{x^\alpha | x^\alpha \notin \langle LT(I)\rangle\}$ son "linealmente independientes módulo I". Es decir,

$$\sum_{\alpha\in\Lambda} c_\alpha x^\alpha = 0 \ mod\ I \Longrightarrow c_\alpha = 0,\ \forall \alpha \in \Lambda\ , \tag{2.9}$$

donde Λ es un subconjunto finito del conjunto $\{\alpha \mid x^\alpha \notin \langle LT(I)\rangle$.

Demostración. 1) Sea G una base de Gröbner de I y sea $f \in k[x_1, ..., x_n]$. Por el algoritmo de la división (Teorema 1.7), el resto $r = \overline{f}^G$ satisface $f = q + r$, donde $q \in I$. Por consiguiente, $f - r = q \in I$, y, por lo tanto $f \equiv r\ mod\ I$. Además, el algoritmo de la división también garantiza que r es combinación k-lineal de monomios $x^\alpha \notin \langle LT(I)\rangle$.

2) Sea $f = \sum_{\alpha\in\Lambda} c_\alpha x^\alpha \equiv 0\ mod\ I$, tenemos $f - 0 = f \in I$. Entonces, si suponemos $f \neq 0$, $LT(f) \in \langle LT(I)\rangle$. Por consiguiente, $LT(f)$ es divisible por algún elemento de $LT(I)$. Pero como $x^\alpha \notin \langle LT(I)\rangle$, es decir, los x^α no son divisibles por ningún elemento de $\langle LT(I)\rangle$, tenemos una contradicción de la cual se sigue que $f = 0$. Es decir, los c_α tienen que ser 0 para todo $\alpha \in \Lambda$. □

Proposición 2.13. Sea $I \subseteq k[x_1, ..., x_n]$ un ideal. Entonces $k[x_1, ..., x_n]/I$ es isomorfo, como k-espacio vectorial, al espacio vectorial generado por $\beta = \{x^\alpha | x^\alpha \notin \langle LT(I)\rangle\}$.

Demostración. Sea $V = k[x_1, ..., x_n]/I$. Por la proposición 2.12, 1, se tiene que β es un sistema generador de V como consecuencia del Teorema de división 1.7. Además, β es un sistema libre, por la proposición 2.12, 2. Luego β es una base de V como k-espacio vectorial.

□

Teorema 2.14 (Teorema de Finitud)**.** *Sea $I \subseteq k[x_1, ..., x_n]$ un ideal. Fijado un orden monomial sobre $k[x_1, ..., x_n]$, las siguientes condiciones son equivalentes:*

1. *Para cada i, $1 \leq i \leq n$, existe un $m_i \geq 0$ tal que $x_i^{m_i} \in \langle LT(I)\rangle$.*

2. *Si G es una base de Gröbner de I, entonces para cada i, $1 \leq i \leq n$, existe un $m_i \geq 0$ tal que $x_i^{m_i} = LT(g)$ para algún $g \in G$.*

3. *El conjunto $\{x^\alpha \mid x^\alpha \notin \langle LT(I)\rangle\}$ es finito.*

4. *El k-espacio vectorial $k[x_1, ..., x_n]/I$ tiene dimensión finita sobre k.*

Además cada una de las anteriores condiciones implican el siguiente enunciado:

5. *La variedad $V(I) \subseteq k^n$ es un conjunto finito.*

De hecho, si k es algebraícamente cerrado, 1-5 son equivalentes.

Definición 2.15. Para cualquier cuerpo k algebraícamente cerrado, por ejemplo $k = \mathbb{C}$, un ideal que satisface cualquiera de las condiciones anteriores se denomina *ideal de dimensión cero.*

Demostración. Vamos a probar que 1-4 son equivalentes probando 1⇔2, 1⇔3 y 1⇔4.

1⇔2) Supongamos que $x_i^{m_i} \in \langle LT(I)\rangle$. Dado que G es una base de Gröbner de I, $\langle LT(I)\rangle = \langle LT(g) \mid g \in G\rangle$. Por la proposición 1.5, existe algún $g \in G$ tal que $LT(g)$ divide a $x_i^{m_i}$. Pero esto implica que $LM(g)$ es una potencia de x_i, como afirmábamos. El recíproco se sigue directamente de la definición de $\langle LT(I)\rangle$ ya que $G \subseteq I$.

1⇔3) Si alguna potencia $x_i^{m_i} \in \langle LT(I)\rangle$ para cada i, entonces los monomios $x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ para algún $\alpha_i \geq m_i$ están todos en $\langle LT(I)\rangle$. Los monomios en el complementario de $\langle LT(I)\rangle$ deben tener $0 \leq \alpha_i \leq m_i - 1$ para cada i. Como resultado, el número de monomios en el complementario de $\langle LT(I)\rangle$ es a lo sumo $m_1 \cdot m_2 \cdots m_n$.

Para el recíproco, supongamos que el complementario tiene $N < \infty$ monomios. Entonces, para cada i, al menos uno de los $N+1$ monomios $1, x_i, x_i^2, ..., x_i^N$ debe estar en $\langle LT(I)\rangle$.

1⇔4) Se sigue de la proposición 2.13.

Para completar la prueba vamos a demostrar que 4⇒5, y, suponiendo que k es algebraícamente cerrado, 5⇒1.

4⇒5) Para demostrar que $V = V(I)$ es finito, es suficiente con demostrar que para cada i existe a lo sumo un número finito de coordenadas i-ésimas distintas para los punto de V.

Sea $P = (p_1, \ldots, p_n) \in V(I)$. Sea $g_i \in G$ tal que $LT(g_i) = x_i^{m_i}$ para $i = 1, \ldots, n$. Entonces p_i es raíz del polinomio de grado a lo más m_i definido por $\hat{g}_i(x_i) = g(p_1, \ldots, p_{i-1}, x_i, p_{i+1}, \ldots, p_n)$. Luego p_i sólo puede tomar un número finito de valores. Fijando cada una de estas posibles raíces, las posibilidades para el resto de p_j para $j \neq i$ también son finitas. Lo que concluye la prueba.

5⇒1) Supongamos que k es algebraícamente cerrado y $V = V(I)$ es finito. Si $V = \emptyset$, entonces $1 \in I$ por 2.10. En este caso, podemos tomar $m_i = 0$ para todo i. Si $V \neq \emptyset$, entonces, para un i fijo, sean $a_1, ..., a_n \in k$ las i-ésimas coordenadas distintas de puntos de V. Formamos el siguiente polinomio de una variable:

$$f(x_i) = \prod_{j=1}^{m}(x_i - a_j).$$

Por construcción, f se anula en todos los puntos de V, así que $f \in I(V)$. Por el Teorema de los ceros, Teorema 2.11, existe algún $N \geq 1$ tal que $f^N \in I$. Pero esto nos dice que el $LM(f^N) \in \langle LT(I)\rangle$. Examinando nuestra expresión de f, vemos que $x_i^{mN} \in \langle LT(I)\rangle$. □

Si k no es algebraicamente cerrado la condición 5 del teorema anterior no es equivalente a las anteriores. Por ejemplo, si $k = \mathbb{R}$, el ideal I de $\mathbb{R}[x, y]$ engendrado por el

polinomio $f(x,y) = x^2 + y^2$ verifica que $V(I) = \{(0,0)\}$, pero para el orden $x < y$, se tiene que $x^m \neq LT(g)$ para g en cualquier base de Gröbner de I.

Ejemplo 2.16. A continuación, vamos a ver un ejemplo de ideal de dimensión 0 en $\mathbb{C}[x,y]$.

Tomamos $I = \langle x, y + x^2, y - x^2 \rangle \subseteq \mathbb{C}[x,y]$. Queremos ver si el conjunto de puntos $V(I) = V(x, y + x^2, y - x^2)$ es un conjunto finito. Este conjunto, está definido por los puntos que cumplen simultáneamente:

$$x = 0,\ y + x^2 = 0,\ y - x^2 = 0$$

Gráficamente, la situación sobre el plano real es la siguiente:

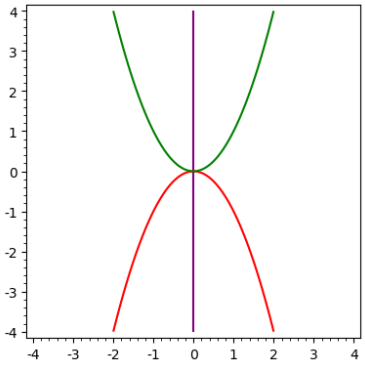

Figura 2.1: Representación gráfica del sistema.

Vemos que, en este ejemplo, la única solución del sistema anterior es el punto $(0,0)$. Es decir, $V(I) = \{(0,0)\} \subseteq \mathbb{C}^2$, que es un conjunto finito. Por consiguiente, el ideal I es un ideal de dimensión 0.

CAPÍTULO 3
Computación simbólica y Machine Learning

3.1. Introducción a Machine Learning

En esta sección vamos a dar una introducción a "Machine Learning" siguiendo las referencias [4, 5].

Como señala [4], gracias a los avances en computación de los últimos años, tenemos la capacidad de guardar cantidades enormes de datos y procesarlos. Además, disponemos de los medios necesarios para acceder a estos datos desde cualquier lugar físico (a través de la red). En todos los ámbitos, se recopilan datos que, posteriormente, son almacenados. Estos datos, una vez almacenados, pasan a ser muy útiles al ser analizados y convertidos en información que puede ser utilizada, por ejemplo, para hacer perfiles sobre clientes, predecir comportamientos futuros, etc.

De hecho, como bien señala [5], todos nosotros generamos datos con la mayoría de acciones que realizamos diariamente: cada vez que compramos un producto, cada vez que vamos al cine, cuando escribimos en un blog, cuando utilizamos redes sociales, etc. estamos generando datos. Y, no solo generamos datos, sino que los consumimos. Queremos productos y servicios especializados para nuestras necesidades.

Por ejemplo, pensemos en una cadena de supermercados que vende sus productos a nivel nacional. Se guardan los detalles de cada transacción: fecha, identificador del consumidor, productos comprados, importe total, etc. Enormes cantidades de datos diarios. Lo que esta cadena de supermercados quiere es ser capaz de predecir qué clientes tienen mayor probabilidad de comprar qué producto, con el objetivo de maximizar su beneficio. Y el cliente quiere encontrar los productos que más se ajustan a sus necesidades. Esta tarea no es sencilla. No sabemos exactamente cuánta gente va a comprarse un libro de cierto autor, ver cierta película, etc. El comportamiento de los clientes cambia con el tiempo y depende del lugar geográfico. Pero, lo que sí sabemos, es que no es completamente aleatorio. Existen ciertos **patrones** en los datos. Para resolver un problema computacionalmente, necesitamos un **algoritmo**, que es una serie de instrucciones que deben ser llevadas a cabo para convertir la entrada en salida. Por ejemplo, un algoritmo para ordenar una lista.

El problema es que para algunas tareas no tenemos un algoritmo. O como en esta memoria los algoritmos son muy costosos computacionalmente y no es "rentable" para las aplicaciones usar algoritmos deterministas. Predecir el comportamiento de los clientes es una de ellas; decidir si un correo electrónico es "spam" o no es otra. No obstante, conocemos la entrada: el correo en cuestión (en el caso más sencillo un archivo de texto). Y, también, conocemos cómo debería de ser la salida: sí/no indicando si el correo es "spam" o no lo es. Lo que se considera "spam" varía con el tiempo y con la pareja de individuos que intercambian el correo. Lo que nos falta de conocimiento, lo compensamos con datos. Fácilmente, podemos conseguir miles de correos de ejemplo, de los cuales sabemos si son "spam" o no. Y, lo que queremos es "aprender" el qué conforma el "spam". Es decir, queremos que un computador extraiga de manera automática el algoritmo que desempeña esta tarea.

Puede que no seamos capaces de identificar el proceso completamente, pero creemos que somos capaces de construir una buena y útil aproximación del mismo. Puede que esta aproximación no sea capaz de explicar todos los datos, pero sí una parte importante de ellos, es decir, sea capaz de detectar ciertos patrones. Estos patrones nos pueden ayudar a entender el proceso, o simplemente podemos utilizarlos para hacer predicciones. Asumiendo que el futuro cercano no será muy diferente del pasado en el que los datos fueron recopilados, se espera que las predicciones futuras sean correctas.

"Machine Learning" es parte de la Inteligencia Artificial (IA). Para ser inteligente, un sistema que se encuentra en un entorno cambiante, tiene que tener la habilidad de aprender, adaptarse a los cambios en el entorno sin que el desarrollador del mismo le de soluciones concretas para sobreponerse a esos cambios. "Machine Learning" nos ayuda a encontrar soluciones a muchos problemas de visión artificial, reconocimiento de voz y robótica. Por ejemplo, reconocimiento facial: es una tarea que hacemos sin mayor esfuerzo. Cada día somos capaces de reconocer a nuestros familiares y amigos al ver sus caras o incluso en fotografías, sin importar sus poses, la luz, su corte de pelo, etc. No obstante, lo hacemos inconscientemente y no sabemos explicar cómo lo hemos hecho, y, por lo tanto, no podemos escribir un algoritmo que desempeñe esta tarea, es decir, tenemos el mismo problema que veíamos con el "spam". Por otro lado, en este caso, lo que sí sabemos es que una cara tiene una estructura. Es simétrica. Tiene ojos, nariz y boca situados en unos lugares determinados. La cara de cada persona tiene un patrón específico compuesto por una combinación de las características anteriores. Analizando ejemplos de imágenes de la cara de una persona, un programa capaz de aprender puede llegar a identificar el patrón específico de la cara de la persona y llegar a reconocerla en otras imágenes. Esto es un ejemplo de *reconocimiento de patrones*.

"Machine Learning" consiste en programar computadores para optimizar un criterio de ejecución mediante datos de ejemplo o experiencia previa [5]. Tenemos un modelo definido con unos parámetros, y, el aprendizaje es la ejecución de un programa informático cuyo objetivo es optimizar los parámetros del modelo utilizando datos de entrenamiento ("training dataset", Apéndice B) o experiencia previa. El modelo puede ser *predictivo* para hacer predicciones futuras, *descriptivo* para extraer conocimiento de los datos, o ambos.

"Machine Learning" utiliza la teoría estadística de construcción de modelos matemáticos, porque la tarea principal es hacer inferencia a partir de los datos de ejemplo. El papel del computador es:

- En el entrenamiento, necesitamos algoritmos eficientes para resolver el problema de optimización de los parámetros. Además de guardar y procesar la enorme cantidad de datos que se van a utilizar.
- Una vez que el modelo ha sido entrenado, su representación y solución algorítmica para hacer la inferencia tienen que ser eficientes.

En algunas aplicaciones, la eficiencia del algoritmo de entrenamiento o del de inferencia (sus complejidades espaciales y temporales) son tan importantes como la precisión de sus predicciones.

A continuación, vamos a ver ejemplos de aplicaciones básicas de "Machine Learning" que tenemos que conocer para comprender el experimento que hemos realizado. Para más información sobre "Machine Learning" ver las fuentes originales [4, 5].

3.2. Ejemplos de aplicaciones de "Machine Learning"

A continuación, listamos algunos ejemplos de aplicaciones de "Machine Learning" siguiendo [5]. No nos centraremos en explicar demasiado todos, simplemente nos centraremos en ver en detalle aquellas aplicaciones que tienen que ver con el experimento que hemos realizado.

Reglas de asociación

Por ejemplo, en una cadena de supermercados, una aplicación de "Machine Learning" es el *análisis de cestas*, que consiste en encontrar relaciones entre productos comprados por clientes. Esto es, si los clientes que normalmente compran X, también compran Y. Si un cliente que compra X no compra Y, entonces es un consumidor de Y en potencia. Una vez encontrado un consumidor en potencia, podemos fijarle con el objetivo de venderle Y.

Para encontrar una *regla de asociación*, nos interesa aprender la probabilidad condicionada $P(Y|X)$ donde Y es el producto que queremos vender y X el producto o conjunto de productos que sabemos que el cliente compra. Si queremos disinguir entre clientes que cumplen un conjunto de atributos D, entonces tenemos que tratar de estimar $P(Y|X, D)$. Este tipo de reglas también son utilizadas para los anuncios que nos aparecen cuando navegamos por Internet.

Clasificación

Por ejemplo, los bancos tratan de predecir el riesgo asociado a los préstamos que van a dar (antes de darlos). Esto es, la probabilidad de impago del importe total de un

cliente, con el objetivo de garantizar beneficios y de no dar préstamos que sobrepasen la capacidad financiera de un cliente.

En puntaje crediticio, el banco calcula el riesgo dados la cantidad del crédito y la información asociada al cliente. La información del cliente incluye datos como: ingresos, ahorros, aval, profesión, edad, historial financiero, etc. El objetivo del banco es inferir una regla general, a partir de los datos, que le permita asociar los atributos de un consumidor dado al riesgo del mismo. Esto es, calibrar un modelo de "Machine Learning" de acuerdo a datos previos (de préstamos anteriores que ha concedido el banco y ya sabe si le han sido devueltos o no, y los atributos del cliente al que se los concedió) para calcular el riesgo de una nueva petición y decidir si lo concede o no dependiendo del resultado.

El ejemplo anterior sería un ejemplo de problema de *clasificación* en el que podrían existir dos clases (aunque en la práctica hay bastantes más): riesgo-bajo y riesgo-alto. El clasificador recibe como *entrada* los datos del cliente, y asigna a la entrada una de las dos clases posibles.

Tras el entrenamiento del modelo, la regla de clasificación tiene la siguiente forma:

$$\texttt{IF } ingresos > \theta_1 \texttt{ AND } ahorros > \theta_2 \texttt{ THEN riesgo-bajo ELSE riesgo-alto} \quad [5]$$

para ciertos valores de θ_1 y θ_2 (que serán determinados en el proceso de entrenamiento del modelo). Esto es un ejemplo de *discriminante*, que es una función que separa los ejemplos en clases diferentes.

Una vez conseguida la regla de clasificación que se ajusta a los datos pasados (y suponiendo que el futuro será parecido al pasado), la aplicación principal es la *predicción*. Esto es, podemos realizar predicciones precisas para nuevas peticiones de préstamos (dados los atributos del cliente, podemos decidir si es de riesgo-bajo o riesgo-alto).

En ciertos casos, en vez de simplemente clasificar en 0/1 (riesgo-bajo/riesgo-alto), nos puede interesar la probabilidad exacta de pertenencia a una de las dos clases. En estos casos, podemos ver la clasificación como una regla de asociación de X (atributos del cliente) a Y (clase de riesgo: 0/1). Ver 3.2. Es decir, dado $X = x$, calcular $P(Y = 1|X = x)$ (o, equivalentemente $P(Y = 0|X = x)$) y a partir de esta probabilidad decidir si concedemos o no el préstamo dependiendo de los posibles beneficios y pérdidas.

La clasificación también se utiliza en *reconocimiento de patrones*: *reconocimiento de escritura*, *reconomiento facial*, *diagnóstico médico*, *reconocimiento del lenguaje*, etc.

Al aprender una regla de clasificación a partir de los datos, también podemos *extraer conocimiento*. Por ejemplo, en nuestro ejemplo de los bancos, a partir del discriminante, podemos saber los atributos que tiene un cliente de riesgo-bajo.

El conocimiento de la regla también nos permite detectar "outliers" (excepciones que no siguen la regla, *anomalías*). En el ejemplo anterior nos podrían llevar a detectar, por ejemplo, casos de fraude.

Nos hemos centrado en explicar en profundidad este apartado ya que nuestro problema pertenece a este tipo de problemas.

Regresión

Supongamos que queremos predecir el precio de coches usados. Las entradas, X, son los atributos del coche: marca, año, motor, etc. La salida, Y, es el precio para dicho coche. Estos problemas, en los que la salida es un valor numérico son problemas de *regresión*. En ellos, el obejetivo del programa de "Machine Learning" es ajustar una función (*función de regresión*) a los datos de entrenamiento. De manera que aprende Y como función de X.

Para más información sobre reglas de asociación, clasificación o regresión ver la fuente original [5].

3.2.1. Aprendizaje

En este apartado vamos a explicar brevemente los tipos principales de aprendizaje siguiendo [5].

Aprendizaje supervisado

Tanto los problemas de clasificación como los problemas de regresión son problemas de este tipo.

Este tipo de problema se generaliza porque existe una entrada X y una salida Y, y la tarea es aprender a mapear la entrada a la salida. "Machine Learning" asume un modelo definido con un conjunto de parámetros. Es decir, $y = g(x|\theta)$, donde g es el modelo y θ son sus parámetros. En el caso de clasificación, y es la codificación de la clase $(0, 1, ..., n)$; en el de la regresión, es un valor númerico. En el caso de clasificación, g es la función discriminante; en el caso de regresión, g es la función de regresión.

El programa de "Machine Learning" optimiza los parámetros, θ, de tal manera que se minimice el error de aproximación. Es decir, que nuestras estimaciones sean lo más cercananas posibles a los valores correctos (dados en el conjunto de entrenamiento).

Aprendizaje no supervisado

En el aprendizaje supervisado existe un "supervisor" que nos proporciona los valores correctos de la salida para ciertos valores de la entrada, y nuestros objetivo es aprender a mapear esas entradas a la salida. En cambio, en el aprendizaje no supervisado no existe este "supervisor", es decir, solo tenemos los datos de entrada. El objetivo de este tipo de aprendizaje es encontrar patrones en los datos de entrada. Los datos de entrada tienen una estructura tal que ciertos patrones ocurren con más frecuencia que otros, y queremos ver qué pasa "generalmente" y qué no lo hace. Esto se conoce como *estimación de densidad*. Un método para desempeñar esta tarea sería el "clustering", el cual consiste en encontrar "clusters" o agrupaciones en los datos de entrada.

Aprendizaje por refuerzo

En algunas aplicaciones, la salida del sistema es una secuencia de *acciones*. En estos sistemas, una sola acción no es importante; lo que es importante es la *política*, que es entendida como la secuencia de acciones correctas que llevan a la consecución del objetivo. No existe una *mejor acción* para un estado intermedio; una acción es buena si es parte de una política buena. Entonces, el programa de "Machine Learning" debe de ser capaz de comprobar la bondad de una política y de aprender de secuencias de acciones buenas para generar una política. Esos métodos de aprendizaje se conocen como algoritmos de *aprendizaje por refuerzo*. Un buen ejemplo de este tipo de aprendizaje se da en *juegos* y en sistemas de navegación (para robots, coches autónomos, etc.).

En nuestro experimento utilizamos aprendizaje supervisado. Para más información sobre los tipos principales de aprendizaje ver la fuente original [5].

CAPÍTULO 4
Bases de Gröbner y Machine Learning

En este capítulo incluímos el desarrollo del experimento realizado para decidir la rentabilidad de calcular una base de Gröbner de un ideal I dado por una familia finita de polinomios $f_1, \dots f_\kappa$ en dos variables x, y. El cuerpo tomado para la experimentación ha sido el de los números racionales, $k = \mathbb{Q}$ para no introducir complejidad asociada a la computabilidad de k. Se trataría de determinar con un "test"si merece la pena calcular una base de Gröbner para calcular los puntos de corte de las curvas planas dadas por $f_1 = 0, \dots, f_\kappa = 0$. Desde el punto de vista del Álgebra computacional, este problema estaría bien resuelto si tuviéramos codificados los puntos de la siguiente manera:

$$(a, b) \in V(I) \text{ si y sólo si } \exists g_0 \in \mathbb{Q}[t], g_i \in \mathbb{Q}[x, y], \text{ con } g_0(a) = 0, g_i(a, b) = 0.$$

Este ideal tendría dimensión 0 siempre que g_0 no fuera el polinomio nulo. Este hecho ha sido tomado como criterio de rentabilidad del test (Definición 4.1) e ilustrado con un ejemplo (Ejemplo 4.2).

Tratamos de diseñar un experimento con vista poder calcular una descomposición algebraica cilíndrica (CAD) en el sentido expuesto en el libro de M. Coste, [7], asociada a una mejor presentación del ideal I. Entonces, siguiendo las ideass expuestas en el interesante trabajo de Huang y colaboradores, [11], nos planteams usar técnicas de "Machine Learnig" para abordar este problema de decisión, que teniendo una fundamentación algebraica, rebasa los límites de las matemáticas debido a las enormes aplicaciones que tiene el diseño asistido por ordenador.

El cálculo de una base de Gröbner, 1.11, en algunas ocaciones, puede conllevar cálculos computacionalmente costosos, y la base de Gröbner obtenida puede no ser de gran ayuda para resolver el problema en cuestión. Por esto creemos que el experimento planteado tiene una gran potencialidad. Aún más si logramos describirlo para ideales de $\mathbb{Q}[x, y, z]$; incluímos más detalle sobre esta cuestión en la sección 4.3. Dado un cuerpo k de característica 0, y una familia finita de polinomios $f_1, \dots, f_\kappa$ en $k[x, y]$, denotaremos por G una base de Gröbner del ideal $I = \langle f_1, \dots, f_\kappa \rangle$. El criterio de rentabilidad de G se ajusta a la siguiente definición.

Definición 4.1. Fijado un orden monomial sobre $k[x, y]$, sea $G = \{g_1, ..., g_t\}$ una base de Gröbner de un ideal $I \subseteq k[x, y]$ distinto de $\{0\}$. Consideramos los siguientes números naturales

$$n = \#\{g \in G \mid \text{g solo depende de una variable}\}$$
$$m = \#\{g \in G \mid \text{g depende de ambas variables}\}.$$

Decimos que la base de Gröbner G es *rentable* si y solo si $n \geq m$.

Este criterio de rentabilidad es el que emplearemos para el estudio de bases de Gröbner de ideales I de dimensión 0 (véase 2.3). Nos interesa ser capaces de determinar las coordenadas de los puntos que pertenecen a la variedad $V(I)$, en la notación introducida en el capítulo 1, mediante el cálculo de una base de Gröbner, caso de que esto sea rentable. Ilustraremos este hecho con el siguiente ejemplo.

Ejemplo 4.2. Consideremos el anillo de polinomios en dos variables $\mathbb{C}[x, y]$, y el orden lexicográfico a $x > y$. Recordemos que sobre $\mathbb{C}[x, y]$, tal y como veíamos en 2.14, para que un ideal, $I \subseteq \mathbb{C}[x, y]$, sea un ideal de dimensión 0 basta con que cumpla una de las propiedades equivalentes dadas en 2.14. En particular, para este ejemplo nos fijaremos en la propiedad 5 allí demostrada, es decir, que $V(I) \subseteq \mathbb{C}^2$ sea un conjunto finito de puntos de $\mathbb{C}^2$.

Consideremos los polinomios $f_1(x, y) = x^2 + y^2 - 4$ y $f_2(x, y) = \frac{xy}{2} + \frac{y^2}{9} - 1$. Consideramos el ideal I engendrado por ellos, $I = \langle f_1, f_2 \rangle$. Por 2.7, $V(I)$ viene dado por los puntos que cumplen simultáneamente $f_1(x, y) = 0$ y $f_2(x, y) = 0$. Si consideramos los puntos que tienen coordenadas reales, la situación geométrica en la que nos encontramos es la siguiente.

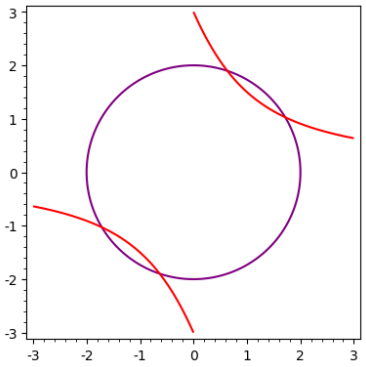

Figura 4.1: Representación gráfica del sistema.

Por consiguiente, y para hacer más sencilla la resolución del sistema que define los puntos de $V(I)$, calculamos una base de Gröbner de I. No entraremos en los detalles del cálculo, pero es fácil ver que

$$G = \left\{ g_1(x, y) = x + \frac{85}{162}y^3 - \frac{20}{9}y,\ g_2(y) = y^4 - \frac{396}{85}y^2 + \frac{324}{85} \right\}$$

es una base de Gröbner de I. Ahora, podemos escribir $V(I) = V(g_1, g_2)$, y el nuevo sistema que obtenemos es el siguiente:

$$x + \frac{85}{162}y^3 - \frac{20}{9}y = 0, \quad y^4 - \frac{396}{85}y^2 + \frac{324}{85} = 0. \tag{4.1}$$

Gráficamente, la situación (sobre el plano real) ahora ha pasado a ser:

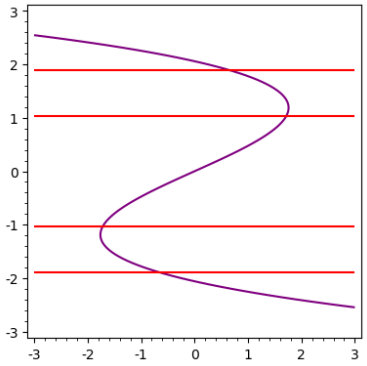

Figura 4.2: Representación gráfica del sistema.

En el sistema (4.1) vemos que los puntos intersección de las curvas han quedado codificados por el polinomio g_2. En efecto, los puntos de $V(I)$ vendrían dados por $P_i = (x_i, y_i)$, con y_i tal que $g_2(y_i) = 0$. Entonces, podemos calcular las raíces del polinomio $g_2(y) = 0$ para obtener los puntos de $V(I)$:

$$P_1 = \left(\sqrt{\frac{2}{5}}, 3\sqrt{\frac{2}{5}}\right), \; P_2 = \left(5\sqrt{\frac{2}{17}}, 3\sqrt{\frac{2}{17}}\right), \; P_3 = -P_2, \; P_4 = -P_1.$$

De hecho, los puntos de $V(I)$ nos dan aun más información. Nos indican que la dimensión como $\mathbb{C}$-espacio vectorial de $\mathbb{C}[x, y]/I$ es 4, véase 2.14.

Hemos podido observar observar en el ejemplo anterior que es mucho más sencillo hallar las soluciones a un sistema del tipo (4.1) que al sistema inicial, ya que tenemos que la segunda ecuación del sistema solo depende de y. Nótese que además, la base de Gröbner G del ejemplo cumple el criterio de rentabilidad que hemos definido en 4.1. Así, el criterio de rentabilidad definido en 4.1 parece "razonable".

4.1. Preparación del experimento

4.1.1. Modelo "*Support Vector Machine*"

El modelo que hemos decidido utilizar ha sido un "*Support Vector Machine*" (SVM) como bien recomienda [11] para estudios del estilo del nuestro.

El SVM para clasificación estándar toma como entrada un conjunto de datos y predice una clase de entre dos posibles de la entrada. Esto lo hace un clasificador

binario no-probabilístico [11]. El SVM representa los "examples", ver Apéndice B, del "training dataset" como puntos en el espacio, mapeados de tal manera que los puntos de distintas categorías estén divididos por un espacio claro, lo más "amplio" posible. El objetivo del SVM es encontrar una superficie de decisión que maximice la distancia a los puntos más cercanos de cada clase pertenecientes al "training dataset". Estos puntos reciben el nombre de "support vectors" [10].

De una manera más formal, sea el conjunto de puntos del "training dataset" $\mathbf{x}_i \in \mathbb{R}^n$, con $i = 1, 2, ..., N$, donde cada $\mathbf{x}_i$ pertenece a una de las dos clases diferentes y su clase correspondiente se denota por $y_i \in \{-1, 1\}$. Asumiendo que los datos son linealmente separables (si no lo son ver [18]), el objetivo del método es separar las dos clases por un hiperplano (superficie de decisión comentada en el párrafo anterior) tal que la distancia a los "support vectors" sea la máxima posible. Un hiperplano tiene la siguiente forma: $\mathbf{w} \cdot \mathbf{x} - b = 0$, donde $\mathbf{w}$ es el vector normal al hiperplano y b es un escalar. Para describir el hiperplano que maximiza la distancia a los "support vectors", utilizamos la siguiente forma:

$$y_i(\mathbf{w} \cdot \mathbf{x_i} - b) \geq 1, \ \ i = 1, ..., l \tag{4.2}$$

Entonces, el hiperplano buscado es aquel que cumple las condiciones (4.2) y minimiza el funcional $\Phi(w) = \frac{1}{2}\|w\|^2$. La minimización se toma respecto al vector $\mathbf{w}$ y al escalar b. Para más información ver [18].

Hallar el hiperplano consiste en resolver este problema de optimización. Se pueden ver detalles sobre su resolución en [18].

Para el caso en el que el problema de clasificación es no-lineal, la idea es mapear cada punto de la entrada a un espacio de mayor dimensión, llamado "feature space". En este espacio, el producto escalar de dos puntos se reemplaza por una función denominada función del núcleo ("kernel function") $K(x, y)$, y se construye el hiperplano de separación. Para más información ver [18, 6]. Este tipo de función aparecerá más adelante en la descripción del experimento en 4.2.

Actualmente, el SVM se utiliza en muchos campos de "Machine Learning", como, por ejemplo, en reconocimiento facial [10].

4.1.2. Implementación

El sistema operativo utilizado ha sido Ubuntu 18.04.4 LTS. Por conveniencia, hemos decidido utilizar SageMath, que es un "software" matemático de código abierto bajo Licencia Pública General. Está construido sobre paquetes de código abierto como NumPy, SciPy, matplotlib, Sympy, Maxima, GAP, FLINT, R y muchos más [2]. El lenguaje de programación que utiliza es un lenguaje basado en Python. En nuestro caso, en Python 2.7.17. Concretamente, hemos utilizado la versión 8.9 de SageMath. En cuanto a los paquetes instalados y sus versiones, ver Apéndice C. El trabajo lo hemos desarrollado sobre "notebooks" de Jupyter.

En cuanto al modelo utilizado, hemos utilizado la clase `SVC` ("Support Vector Classifier"), disponible en `sklearn.svm` ("svm" de "Support Vector Machine"). Para más información ver [3].

Hemos elegido utilizar SageMath por la flexibilidad que nos brinda el lenguaje Python y la posibilidad de programar simbólicamente. No obstante, el rendimiento podría ser superior (sobre todo en cuanto a tiempo) utilizando, por ejemplo, SVM-LIGHT como en [11].

Por último, hablaremos sobre los ficheros que hemos empleado y lo que hemos hecho en cada uno de ellos (solo hablaremos de los ficheros que están relacionados con el experimento, omitiendo los utilizados para generar la lista de paquetes instalados, los gráficos, etc.; si se quieren detalles sobre los comandos utilizados para ello ver Apéndice A).

Antes que nada, vamos a comentar la estructura común a todos los ficheros. Después especificaremos las características particulares de cada uno. Las características generales son: primero, generamos el conjunto de ideales con n generadores por ideal, después, generamos el anillo de polinomios con coeficientes racionales, sobre dos variables y con orden lexicográfico. A continuación, definimos las funciones para generar el conjunto de "features" y la "label" de cada "example", Apéndice B. Una vez definidas, generamos el conjunto de datos final con las "features" y las "labels" a partir del conjunto de ideales generado previamente. Creamos los "arrays" de "Numpy" a partir del conjunto de datos final y los dividimos en "training dataset" y "test dataset", Apéndice B. Finalmente, creamos y entrenamos el modelo, predecimos las "labels" del conjunto "test dataset", y sacamos estadísticas sobre el rendimiento del modelo. Para información sobre los comando utilizados para realizar estas acciones ver Apéndice A.

- *MLPolynomials_deg2.ipynb*, *MLPolynomials_deg3.ipynb*, *MLPolynomials_deg4.ipynb*, *MLPolynomials_deg5.ipynb*: en estos ficheros hemos seguido la estructura antes mencionada utilizando $n = 2$ generadores por ideal de grado máximo 2, 3, 4 y 5, respectivamente.

- *MLPolynomials_deg5_experiment_2gens.ipynb*, *MLPolynomials_deg5_experiment_3gens.ipynb*: en estos ficheros, hemos hecho lo mismo que en *MLPolynomials_deg5.ipynb*, utilizando $n = 2$ y $n = 3$ generadores por ideal, respectivamente, y hemos repetido el experimento m veces (variando m) para comprobar si los resultados se mantenían.

- *MLPolynomials_deg2_to_10_experiment_2gens_GSCV.ipynb*: este es el fichero que contiene el experimento final como tal. En él, hemos puesto en práctica todo lo aprendido en los anteriores. Primero, hemos seguido la estructura principal antes mencionada tomando $n = 2$ generadores por ideal. Después, hemos realizado una búsqueda de los mejores hiperparámetros para el modelo utilizando una búsqueda de rejilla (este tipo de búsqueda prueba todas la combinaciones posibles de los valores de hiperparémetros que le pasamos). Para las combinaciones de valores para los hiperparámetros hemos tomado como idea [11], pero las hemos modificado de acuerdo a los resultados que hemos ido obteniendo y a la capacidad computacional disponible. Finalmente, hemos predicho con el modelo con los hiperparámetros optimizados y hemos calculado estadísticas.

4.1.3. "Dataset", "features" y "label"

Como bien explica [11], es evidente la ausencia de conjuntos de datos fiables y mantenidos de Álgebra Computacional relacionados con nuestro problema y similares (ni siquiera ellos han publicado el conjunto de datos que han utilizado). No obstante, en [11] han tomado la decisión de generar su propio conjunto de datos para el problema, y han explicado paso a paso cómo han generado sus datos.

Así que, siguiendo su ejemplo y explicaciones, hemos decidido generar nuestro propio conjunto de datos. Aunque la forma de generarlos no ha sido la misma, ya que nosotros hemos utilizado SageMath en vez de Maple, y hemos modificado el problema de la manera que exponemos a continuación.

En [11] utilizan polinomios con 3 variables x, y, z con el orden lexicográfico ($x < y < z$); mientras que nosotros utilizamos polinomios con 2 variables x, y con el orden lexicográfico ($x < y$). Ellos restringen el número de términos por polinomio a 2 y los coeficientes a un intervalo; nosotros no restringimos el número de términos y permitimos cualquier racional como coeficiente, lo que es suficiente para nuestro experimento. Por último, ellos varían el grado total entre 2, 3 y 4; nosotros lo hacemos desde grado total 1 a 10 (ambos extremos incluidos).

La relajación del problema reside en la reducción del problema a 2 variables. No obstante, esta ha sido necesaria debido a la capacidad computacional de la que disponemos (un ordenador personal).

Cumpliendo las características que hemos listado antes, hemos generado un conjunto de 10000 muestras, donde cada muestra es el conjunto de generadores de un ideal. Es decir, hemos generado aleatoriamente polinomios con las características listadas anteriormente de manera que cada n polinomios (donde n es el número de generadores del ideal y varía entre ficheros como hemos visto en 4.1.2) forman una muestra.

Los polinomios han sido generados con el método `random_element()` que se invoca sobre un `PolinomialRing` y se le puede especificar un grado máximo como argumento (el cual hemos ido variando).

Las características ("features") y etiquetas ("labels") utilizadas

A partir del conjunto de datos, generado como hemos especificado en 4.1.3, hemos generado las "features" de cada muestra y la hemos etiquetado, dando lugar a los "examples". Las "features" iniciales las hemos obtenido tomando algunas de [11]. No obstante, las hemos ido variando, de acuerdo a lo que hemos creído razonable, hasta que hemos conseguido el conjunto de "features" final. Este conjunto final es el que presentamos a continuación.

Las "features" que hemos empleado han sido:

- Número de polinomios homogéneos entre los generadores.
- Diferencia total entre el grado total de los generadores. Calculada dos a dos entre los generadores y sumados todos los resultados.

- Diferencia total entre el número de términos de los generadores. Calculada dos a dos entre los generadores y sumados todos los resultados.
- Número de términos que dependen de x. Se cuentan entre todos los generadores.
- Número de términos que dependen de y. Se cuentan de entre todos los generadores.
- Diferencia entre el número de términos que depende de x y el número de términos que depende de y.
- Número de componentes homogéneas. Se cuentan entre todos los generadores.

En cuanto a la "label", simplemente hemos calculado la base de Gröbner y hemos comprobado si se cumplía el criterio de "rentabilidad" 4.1. Si se cumplía hemos etiquetado el "example" con `True` (podemos pensarlo como 1); si no con `False` (podemos pensarlo como 0).

4.2. Experimento

En este apartado, nos centraremos en explicar el experimento final. Es decir, nos vamos a centrar en *MLPolynomials_deg2_to_10_experiment_2gens_GSCV.ipynb*, concretamente, en el trabajo llevado acabo en ese fichero 4.1.2. No obstante, en las conclusiones hablaremos de los resultados más destacables de algunos de los otros ficheros.

Descripción

El principal de este experimento es crear un modelo capaz de decidir si el cálculo de una base de Gröbner será rentable de acuerdo al criterio establecido en 4.1.

Una vez etiquetado el conjunto de datos (generado como explicamos en 4.1.3) de la manera especificada en 4.1.3 hemos obtenido que 5200 de los 10000 "examples" pertenecen a la clase `True` y el resto a la clase `False`. Es decir de las 10000 bases de Gröbner, 5200 serían rentables de acuerdo al criterio definido en 4.1. Después, hemos dividido el conjunto de datos en dos: 80 % para "train" y 20 % para "test", Apéndice B. A continuación, hemos procedido a buscar los mejores hiperparámtros para el modelo utilizando el conjunto de "train" y, posteriormente, hemos tratado de predecir las "labels" del conjunto de "test". Las "features" y las "labels" son las especificadas en 4.1.3 y la estructura del experimento es la correspondiente a *MLPolynomials_deg2_to_10_experiment_2gens_GSCV.ipynb* explicada en 4.1.2.

En cuanto a los valores de los hiperparámetros para la búsqueda en rejilla, señalamos que

- Hemos utilizado un `kernel` RBF ("Radial Basis Function") siguiendo las indicaciones de [11].

- Para los valores de γ hemos utilizado los mejores obtenidos en búsquedas previas al experimento final a la ejecución final. De ahí las variaciones que podemos observar. $\gamma = \{2e{-}10, 2e{-}9, 2e{-}8, 2e{-}7, 2e{-}6, 2e{-}5, 1e{-}3, 1e{-}2, 1e{-}1, 1\}$.
- Para los valores de C hemos seguido el mismo procedimiento que para los de γ. $C = \{1e{-}1, 1, 1e1, 1e2, 2e5, 2e7, 2e8, 2e9, 2e10\}$.

También, comentar que el modelo generado para combinación de hiperparámetros ha sido entrenado sobre 3 conjuntos de validación cruzada ("cross validation", Apéndice B) formados a partir del 80 % de los datos destinados a "train". Para más información ver [3].

Las métricas para medir el rendimiento

Por último, vamos a explicar las métricas que utilizamos para medir el rendimiento del modelo. Sea tp el número de verdaderos positivos, fp el de falsos positivos y fn el de falsos negativos. Se definen "Precision", "Recall" y "F1-Score" (en el caso de clasificación binaria) como sigue:

$$\text{Precision} = \frac{tp}{tp + fp}, \text{ Recall} = \frac{tp}{tp + fn}, \text{ F1-score} = 2\frac{precision \cdot recall}{precision + recall}.$$

"Precision" mide la habilidad del modelo para no etiquetar como `True` una muestra de la clase `False`. "Recall" mide la habilidad del modelo para encontrar la clase `True`. Y, "F1-Score" es una media armónica de "Precision" y "Recall". Además de todas las anteriores, también generamos la matriz de confusión, que nos permite ver los verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos.

Resultados

Vamos a ver los resultados obtenidos para cada parte.

La mejor combinación de hiperparámetros ha resultado ser $\gamma = 1e{-}1$ y $C = 1$. El resto de parámetros son los que utiliza el modelo por defecto. Podemos verlos en [3]. Para estos hiperparámetros, los resultados obtenidos han sido los siguientes:

- *Matriz de confusión*:

	False	True
False	608	339
True	305	748

Tabla 4.1: Matriz de confusión para el modelo.

Si denominamos M a la matriz de confusión, por definición [3], M es tal que M_{ij} es igual al número de observaciones que sabemos que están en la clase i y se les ha predicho la clase j. Es decir, para la clase `False` hemos acertado

608 (verdaderos negativos) predicciones y fallado 339 (falsos negativos); para la clase `True` hemos acertado 748 (verdaderos positivos) predicciones y fallado 305 (falsos positivos). El número total de ejemplos en el "test" es 2000 (la suma de todos), y de ellos hemos acertado 1356. Es decir, el 67.8 %.

- *Informe de clasificación*:

	Precision	Recall	F1-score	Support
False	0.67	0.64	0.65	947
True	0.69	0.71	0.70	1053
micro avg	0.68	0.68	0.68	2000
macro avg	0.68	0.68	0.68	2000
weighted avg	0.68	0.68	0.68	2000

Tabla 4.2: Informe de clasificación para el modelo.

Las métricas que presenta la tabla (4.2) son las explicadas en la sección 4.2. No obstante, destacar que el informe añade las mismas métricas calculadas para la clase `False`. Estas se calculan sustituyendo verdaderos positivos por verdaderos negativos, y falsos positivos por falsos negativos y viceversa (en las fórmulas presentadas en 4.2). También calcula varias medias (no entraremos en los detalles de cada una) que, dado que estamos en clasificación binaria, coinciden con la media aritmética. Finalmente, el "Support" representa el número de "examples" en cada clase del "test dataset". Para más información ver [3].

En cuanto a los resultados del informe, podemos ver que el modelo tiene más "Precision" y "Recall" prediciendo la clase `True` que la `False`: 0.69 y 0.71 frente a 0.67 y 0.64, respectivamente. Esto es interesante dado que lo que nos interesa es ser capaces de predecir esa clase.

4.3. Conclusiones y trabajo futuro

Se ha planteado el problema de la rentabilidad del cómputo de una base de Gröbner para un ideal I del anillo de posinomios $\mathbb{Q}[x, y]$. El criterio de rentabilidad establece que deseamos encontrar bases de Gröbner asociadas a un conjunto finito de puntos. Si estos puntos estuvieran dados a priori Möller y Buchberger, [15], demuestra que calcular una tal base es un problema de complejidad exponencial. Otra opción, es contemplar la utilidad de este cálculo fijada una aplicación concret. Esta ha sido nuestra opción guiados por los trabajos de Huang y colaboradores, [11]. Nos hemos centrado por esto en criterios que garanticen su utilidad para realizar CAD en dimensión 2.

Lo primero que destacar en nuestro experimento, es mencionar que en las pruebas que hemos realizado con más de dos generadores, el porcentaje de bases de Gröbner rentables de acuerdo al criterio 4.1 subía demasiado. Por ejemplo, para 3 generadores pasaba a estar en torno al 94.68 %. Por lo tanto, el modelo siempre predecía la clase `True`. Por ello, decidimos fijar el número de generadores en 2. Para 2 generadores,

el porcentaje está en torno al 52 %. También hemos probado a variar el grado de los polinomios, lo cual ha producido el mismo efecto que aumentar el número de generadores. Es por ello, que finalmente decidimos generar polinomios variando el grado desde 2 hasta 10.

En cuanto a los resultados obtenidos, hemos obtenido un porcentaje de acierto global del 67.8 %, mientras que en [11] han conseguido un porcentaje de acierto de, en media, en torno al 76.34 %. No obstante, nuestro resultado ha sido conseguido con una capacidad computacional baja que no nos ha permitido optimizar los parámetros tanto como nos hubiera gustado, ni realizar muchas más pruebas que propondremos como trabajo futuro. Volviendo a los resultados obtenido, y como ya adelantábamos en 4.2, destacar que ese 71 % de "Recall" en el modelo es un resultado positivo, ya que nos dice que lo que mejor hace el modelo es predecir la clase `True`. Lo cual es el objetivo principal del experimento. Luego, podemos concluir que el modelo desarrollado cumple con su objetivo principal en un 71 % de los casos generados aleatoriamente y con un porcentaje de acierto global del 67.8 %. No obstante, creemos que estos porcentajes son mejorables, y por ello proponemos a continuación ciertas líneas de trabajo futuro.

Desde el puntos de vista informático, nos gustaría avanzar en las siguientes líneas.

1. Emplear otros lenguajes de programación para intentar conseguir un mejor rendimiento, como comentábamos en 4.1.2.
2. Utilizar librerías que permitan paralelizar la búsqueda en rejilla para optimizar los hiperparámetros del modelo.

En cuanto al trabajo futuro con un contenido mayor de matemáticas, proponemos las siguientes líneas de trabajo principales:

- Probar otros modelos de "Machine Learning", ya que en nuestro caso nos hemos centrado en el modelo utilizado por [11] debido al tiempo y capacidad computacional disponible. En la actualidad han aparecido muchos nuevos modelos y muy potentes; capaces de utilizar, por ejemplo, transferencia de aprendizaje. Esta ténica consiste en entrenar el modelo sobre conjuntos de datos enormes y luego entrenar al modelo sobre conjuntos de datos de problemas particulares [16]. Creemos que podría llegar a ser aplicada a problemas de este tipo.
- Ampliar el experimento a dimensión 3. Esto es, utilizar polinomios en tres variables. Podría ser aplicado de manera bastante eficaz para "Cylindrical Algebraic Decomposition" (CAD) de conjuntos semialgebraicos en $\mathbb{R}^3$.
- En nuestro trabajo, surge de manera natural el problema, nada trivial, de calcular las raíces de un polinomio en una variable. Es un problema fundamental y estudiado durante mucho tiempo en Álgebra computacional, con aplicaciones en muchas otras áreas. En la actualidad se usan algoritmos simbólico -numéricos para el tratamiento de polinomios con coeficientes computables, véase [17, 12, 13]. Estos trabajos incorporados a los resultados expuestos en esta memoria podrían proporcionar un enfoque *posibilista* a la CAD para regiones semialgebraicas del plano real.

Bibliografía

[1] *Sagemath documentation*. https://doc.sagemath.org/, 2020.

[2] *SageMath Mathematical Software System - Sage*. https://www.sagemath.org/, 2020.

[3] *scikit-learn: machine learning in Python — scikit-learn 0.22.2 documentation*. https://scikit-learn.org/stable/, 2020.

[4] E. Alpaydin, *Introduction to machine learning*, Adaptive computation and machine learning, MIT Press, Cambridge, Mass, 2004. OCLC: ocm56830710.

[5] ——, *Introduction to machine learning*, Adaptive computation and machine learning, The MIT Press, Cambridge, Massachusetts, third edition ed., 2014.

[6] B. E. Boser, I. M. Guyon, and V. N. Vapnik, *A training algorithm for optimal margin classifiers*, in Proceedings of the fifth annual workshop on Computational learning theory, 1992, pp. 144–152.

[7] M. Coste, *An introduction to semialgebraic geometry*, RAAG network school, 145 (2002), p. 30.

[8] D. A. Cox, J. Little, and D. O'Shea, *Using Algebraic Geometry*, vol. 185, Springer-Verlag, GTM, Berlin, Heidelberg, 2005.

[9] ——, *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*, Springer-Verlag, UTM, Berlin, Heidelberg, 2007.

[10] B. Heisele, P. Ho, and T. Poggio, *Face recognition with support vector machines: Global versus component-based approach*, in Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001, vol. 2, IEEE, 2001, pp. 688–694.

[11] Z. Huang, M. England, D. J. Wilson, J. Bridge, J. H. Davenport, and L. C. Paulson, *Using machine learning to improve cylindrical algebraic decomposition*, Mathematics in Computer Science, 13 (2019), pp. 461–488.

[12] A. Kobel, F. Rouillier, and M. Sagraloff, *Computing real roots of real polynomials ... and now for real!*, in Proceedings of the ACM on International

Symposium on Symbolic and Algebraic Computation, ISSAC '16, New York, NY, USA, 2016, Association for Computing Machinery, p. 303–310.

[13] ——, *Computing real roots of real polynomials ... and now for real!* http://anewdsc.mpi-inf.mpg.de/, 2016.

[14] R. Laubenbacher and B. Stigler, *A computational algebra approach to the reverse-engineering of gene regulatory networks*, J. Theor. Biol., 229 (2004), pp. 523—-537.

[15] H. Möller and B. Buchberger, *The construction of multivariate polynomials with preassigned zeros*, Lecture Notes in Computer Science. In: Calmet J. (eds) Computer Algebra. EUROCAM 1982., 144 (2005).

[16] S. J. Pan and Q. Yang, *A survey on transfer learning*, IEEE Transactions on knowledge and data engineering, 22 (2009), pp. 1345–1359.

[17] M. Sagraloff and K. Mehlhorn, *Computing real roots of real polynomials*, Journal of Symbolic Computation, 73 (2016), pp. 46–86.

[18] V. N. Vapnik, *An overview of statistical learning theory*, IEEE transactions on neural networks, 10 (1999), pp. 988–999.

APÉNDICE A

Comandos de SageMath utilizados en la memoria

A continuación, presentamos la lista de los comandos principales de SageMath utilizados para la memoria:

- `PolynomialRing(QQ, n)`: genera un anillo de polinomios con coeficientes sobre "QQ" y "n" variables. Como parámetro opcional se puede añadir un orden monomial con `order`.
- `ideal(f, g)`: ideal generado por "f" y "g".
- `I.groebner_basis()`: devuelve una base de Gröbner para el ideal "I".
- `spol(f,g)`: calcula el S-polinomio de "f" y "g" 1.12.
- `(f).reduce(B)`: reduce el polinomio "f" respecto a la base "B".
- `I.vector_space_dimension()`: devuelve la dimensión del espacio vectorial del anillo módulo el ideal "I". Si el ideal no es de dimensión 0 devuelve "+Infinity".
- `f.degree(x)`: devuelve el grado de "x" en el polinomio "f".
- `f.lt()`: devuelve el "leading term" del polinomio "f".
- `implicit_plot(f, xrange, yrange)`: genera la gráfica de $f(x_1, x_2) = 0$ ("f" es una función que tiene que depender de dos variables) en el rango de x y de y especificados. Como parámetro opcional se puede introducir, entre otros, un color con `color`.
- `solve(listofeqs, x, y)`: resuelve el sistema de ecuaciones dado por una lista de ecuaciones ("listofeqs") sobre las variables que aparecen a continuación.
- `random.seed(n)`: genera una semilla aleatoria a partir de "n" para los funciones "aleatorias".
- `f.exponents()`: devuelve los exponentes de los monomios que aparecen en "f".

- `randint(n, m)`: genera un entero aleatorio entre "n" y "m" (ambos extremos incluidos).
- `P.random_element()`: devuelve un polinomio aleatorio perteneciente al anillo de polinomios "P". Tiene argumentos opcionales como `degree` para especificar el grado máximo del polinomio.
- `f.is_homogeneous()`: devuelve "True" si "f" es homogéneo y "False" en caso contrario.
- `numpy.array(item)`: crea un array de "numpy" a partir de un "item" que puede ser, entre otras cosas, una lista.
- `sklearn.model_selection.train_test_split(X, y)`: genera los subconjuntos "training dataset" y "test dataset" a partir de las "features" ("X") y las "labels" ("y"). Es muy flexible y admite muchos parámetros extra como `test_size` para indicar el porcentaje de datos que se quiere utilizar como "test".
- `sklearn.svm.SVC()`: devuelve un modelo "Support Vector Machine" para clasificación. Se pueden añadir muchos parámetros opcionales para definir el modelo. Por ejemplo, `C`, `gamma` o `kernel`.
- `classifier.fit(X_train, y_train)`: entrena al modelo "classifier". "X_train" son las "features" e "y_train" son las "labels" correspondientes.
- `classifier.predict(X_test)`: devuelve las "labels" predichas para las "features" recibidas como argumento ("X_test").
- `sklearn.model_selection.GridSearchCV(model, param_grid)`: invoca el método `fit()` sobre el modelo "model" variando los hiperparámetros con los recibidos en "param_grid" utilizando validación cruzada ("cross validation", de ahí el CV del final). Es muy flexible y admite muchos parámetros opcionales. Finalmente, devuelve el modelo con los hiperparámetros que mejor resultado han dado.
- `grid.best_estimator_`: imprime la información del mejor estimador. Se invoca sobre el retorno de `GridSearchCV()`.
- `sklearn.metrics.classification_report(y_test, preds)`: imprime un informe haciendo comparativas entre las "labels" reales ("y_test") y las predichas por el modelo ("preds").
- `sklearn.metrics.confusion_matrix(y_test, preds)`: imprime la matriz de confusión asociada a las clases dadas las "labels" reales ("y_test") y las predichas por el modelo ("preds").
- `sage.misc.package.list_packages()`: lista todos los paquetes de SageMath con información extra como sus versiones o si están o no instalados.
- `sklearn.show_versions()`: devuelve la versión de "sklearn" y de todos los paquetes de los que depende.

APÉNDICE B
Glosario "Machine Learning"

A continuación, presentamos una lista de conceptos (definido de manera breve y al nivel que necesitamos en este trabajo) importantes para "Machine Learning":

- **"Features"**: son las características que funcionan como entrada del algoritmo de ML, conforman un vector numérico (cuya dimensión depende del número de "features").
- **"Label"**: es la clase asociada a un vector de "features". Funciona como salida del algoritmo de ML.
- **"Example"**: muestra que consta del vector de entradas ("features") y la salida correspondiente a las mismas ("label"). Corresponde a una fila en la tabla de datos.
- **"Training dataset"**: conjunto de datos utilizados para entrenar el modelo de ML. Este conjunto está formado por varios "examples".
- **"Train"**: consiste en entrenar el modelo, para ello, el modelo recibe las "features" y la "label" que se corresponde a esas "features". De esta forma, el modelo "aprende".
- **"Test dataset"**: conjunto de datos utilizados para comprobar el funcionamiento del modelo de ML. Este conjunto también está formado por varios "examples". Normalmente, el conjunto de datos con todos los "examples" se divide en los conjuntos "train data" y "test data" (aunque puede haber variaciones como conjuntos de validación, etc.).
- **"Test"**: dado un conjunto de "examples", consiste en que el modelo prediga la "label" dadas las "features", y comprobar, posteriormente, si la "label" se corresponde con la que tenía ese "example". Es decir, comprobamos si el modelo el funcionamiento del modelo.
- **"Data to predict"**: conjunto de datos sobre los que queremos realizar predicciones. De estos datos solo conocemos las "features". El objetivo principal del modelo que construimos a partir del algoritmo, es predicir este conjunto correctamente.

- **"Cross validation"**: consiste en particionar los datos de entrenamiento en distintas particiones. A continuación se entrena el modelo sobre cada una de ellas y se calcula la media aritmética de la evalución del modelo sobre cada una de las particiones. Se garantizar que el rendimiento del modelo es independiente de la partición.

APÉNDICE C
Paquetes instalados en SageMath

A continuación vamos a indicar cómo instalar `sklearn` en SageMath, y después listaremos todos los paquetes que tenemos instalados en SageMath con su correspondiente versión.

Para instalar `sklearn` en SageMath hay que instalar también todas sus dependencias. Para ello basta con (al menos en Ubuntu 18.04.4 LTS) ejecutar en una celda de Jupyter con el kernel de SageMath el siguiente comando: `!sage --pip install numpy scipy sklearn pandas`.

En cuanto a los paquetes instalados y sus versiones son los siguientes:

- alabaster: 0.7.12
- appnope: 0.1.0.p0
- arb: 2.16.0.p0
- babel: 2.6.0
- backports_abc: 0.5
- backports_functools_lru_cache: 1.5
- backports_shutil_get_terminal_size: 1.0.0.p1
- backports_ssl_match_hostname: 3.5.0.1.p0
- bleach: 3.1.0
- boost_cropped: 1.66.0.p0
- brial: 1.2.5
- bzip2: 1.0.6-20150304.p0
- cddlib: 0.94j
- certifi: 2019.3.9

- cliquer: 1.21.p4
- combinatorial_designs: 20140630.p0
- configparser: 3.7.4
- conway_polynomials: 0.5
- curl: 7.62.0.p0
- cvxopt: 1.1.8.p2
- cycler: 0.10.0.p0
- cypari: 2.1.1
- cysignals: 1.10.2
- cython: 0.29.12.p0
- dateutil: 2.5.3
- decorator: 4.4.0
- defusedxml: 0.6.0
- docutils: 0.14
- ecl: 16.1.2.p5
- eclib: 20190226
- ecm: 7.0.4.p1
- elliptic_curves: 0.8.1
- entrypoints: 0.3
- enum34: 1.1.6
- fflas_ffpack: 2.4.3
- flask: 0.10.1.p0
- flask_autoindex: 0.6.p0
- flask_babel: 0.9.p0
- flask_oldsessions: 0.10.p0
- flask_openid: 1.2.5.p0
- flask_silk: 0.2.p0
- flint: 2.5.2.p4

- flintqs: 1.0.p0
- fplll: 5.2.1
- fpylll: 0.4.1dev
- functools32: 3.2.3-2.p0
- future: 0.17.1
- gap: 4.10.2.p1
- gc: 7.6.4.p0
- gcc: 7.4.0
- gf2x: 1.2.p0
- gfan: 0.6.2.p1
- giac: 1.5.0.63.p0
- givaro: 4.1.1
- glpk: 4.63.p2
- gmpy2: 2.1.0b1
- graphs: 20161026.p0
- gsl: 2.5
- html5lib: 1.0.1
- imagesize: 1.1.0
- iml: 1.0.4p1.p2
- ipaddress: 1.0.22
- ipykernel: 4.8.2
- ipython: 5.8.0
- ipython_genutils: 0.2.0
- ipywidgets: 7.4.2
- itsdangerous: 1.1.0
- jinja2: 2.10
- jmol: 14.29.52
- jsonschema: 2.6.0

- jupyter_client: 5.2.4
- jupyter_core: 4.4.0
- kiwisolver: 1.0.1
- lcalc: 1.23.p19
- libatomic_ops: 7.6.2
- libbraiding: 1.0
- libffi: 3.2.1
- libgd: 2.1.1.1.p1
- libhomfly: 1.02r4
- linbox: 1.6.3
- lrcalc: 1.2.p1
- m4ri: 20140914.p0
- m4rie: 20150908.p0
- markupsafe: 1.1.0
- mathjax: 2.7.4.p0
- matplotlib: 2.2.4.p0
- maxima: 5.42.2
- mistune: 0.8.4
- mpc: 1.1.0
- mpfi: 1.5.2
- mpfr: 4.0.1.p0
- mpir: 3.0.0-644faf502c56f97d9accd301965fc57d6ec70868.p0
- mpmath: 1.1.0
- nauty: 26r1.p0
- nbconvert: 5.4.0
- nbformat: 4.4.0
- ncurses: 6.0.p0
- networkx: 2.2

- nose: 1.3.7
- notebook: 5.7.6.p0
- ntl: 11.3.2
- numpy: 1.16.1
- openblas: 0.3.6.p0
- packaging: 18.0
- palp: 2.1.p2
- pandas: 0.24.2
- pandocfilters: 1.4.2
- pari: 2.11.1.p2
- pari_galdata: 20080411.p0
- pari_seadata_small: 20090618.p0
- pathlib2: 2.3.3
- pathpy: 7.1.p0
- pcre: 8.40.p2
- pexpect: 4.6.0.p0
- pickleshare: 0.7.5
- pillow: 5.3.0.p0
- pip: 18.1
- pkgconfig: 1.4.0
- planarity: 3.0.0.5.p0
- polytopes_db: 20170220.p0
- ppl: 1.2.p1
- pplpy: 0.8.4
- prometheus_client: 0.5.0
- prompt_toolkit: 1.0.15
- psutil: 5.2.0.p2
- ptyprocess: 0.5.1.p0

- pycygwin: 0.1
- pygments: 2.3.1.p0
- pynac: 0.7.26
- pyparsing: 2.3.0
- python2: 2.7.15.p1
- python3: 3.7.3.p1
- python_openid: 2.2.5.p0
- pytz: 2018.7
- pyzmq: 18.1.0
- r: 3.6.1
- ratpoints: 2.1.3.p5
- readline: 6.3.008.p0
- requests: 2.13.0
- rpy2: 2.8.2.p1
- rubiks: 20070912.p21
- rw: 0.7.p0
- sagenb: 1.1.2
- sagenb_export: 3.2
- sagetex: 3.3
- scandir: 1.9.0
- scipy: 1.2.0
- send2trash: 1.5.0
- setuptools: 40.6.3
- setuptools_scm: 3.1.0
- simplegeneric: 0.8.1.p0
- singledispatch: 3.4.0.3.p0
- singular: 4.1.1p2.p0
- six: 1.12.0

- sklearn: 0.20.4
- snowballstemmer: 1.2.1.p0
- speaklater: 1.3.p0
- sphinx: 1.8.5.p0
- sphinxcontrib_websupport: 1.1.0
- sqlite: 3270100
- subprocess32: 3.5.3
- symmetrica: 2.0.p11
- sympow: 1.018.1.p13
- sympy: 1.4.p0
- tachyon: 0.98.9.p7
- terminado: 0.8.1
- testpath: 0.4.2
- thebe: 9624e0a0.p0
- threejs: r105
- tornado: 4.5.2
- traitlets: 4.3.2
- twisted: 16.3.0.p0
- typing: 3.6.6
- vcversioner: 2.16.0.0.p0
- wcwidth: 0.1.7.p0
- webencodings: 0.5.1
- werkzeug: 0.14.1
- widgetsnbextension: 3.4.2
- xz: 5.2.2.p0
- yasm: 1.3.0.p0
- zeromq: 4.2.5
- zn_poly: 0.9.1.p0
- zope_interface: 4.6.0