Sección de Matemáticas
Universidad de La Laguna

Daniel Jaén Guedes

Introducción al Algebraic Machine Learning

Introduction to Algebraic Machine Learning

Trabajo Fin de Grado
Grado en Matemáticas
La Laguna, Junio de 2023

Dirigido por
María del Socorro García Román

María del Socorro García Román
Departamento de Matemáticas,
Estadística e Investigación
Operativa
Universidad de La Laguna
38200 La Laguna, Tenerife

Agradecimientos

Quiero agradecer a mi familia que siempre ha estado ahí para apoyarme a lo largo de mis estudios, además de a los amigos que he hecho en estos, ya que sin su apoyo y ayuda quizás no estaría presentando este trabajo ahora.

También agradecer a mi tutora María del Socorro García Román, por su ayuda en este trabajo, y sobre todo, por ofrecerme la oportunidad de investigar este tema.

Autor del TFG
La Laguna, 22 de mayo de 2023

Resumen · Abstract

Resumen

El Machine Learning (ML), dentro de la Inteligencia Artificial, nos permite aprender características y poder predecir el valor de variables para las cuales sólo tenemos ejemplos. Varios de los métodos clásicos de ML suelen basarse en la minimización de funciones para ajustar una gran cantidad de parámetros en un modelo creado aleatoriamente. Este método presenta varios problemas, como puede ser la dificultad en encontrar mínimos de funciones complicadas y con muchas variables, o que en ocasiones aparece el denominado "overfitting", es decir, el modelo se ajusta demasiado a los datos de entrenamiento y no generaliza bien.

En este trabajo vamos a exponer la alternativa planteada en [10], denominada Algebraic Machine Learning (AML), basada en estructuras algebraicas, principalmente grafos y semirretículos, para encontrar un modelo predictivo, que soluciona los problemas anteriores, y además es libre de parámetros.

En este trabajo vamos a estudiar su funcionamiento como algoritmo de clasificación, es decir, orientado a obtener un modelo que nos permita predecir si un elemento posee una cierta propiedad. Explicaremos cómo encontrar una representación algebraica de los elementos a clasificar respecto de un conjunto especial de elementos ("átomos"), para luego ver paso a paso el algoritmo y cómo se aplica al problema de identificación de barras negras verticales en cuadrículas 2x2 de píxeles blancos o negros.

Mediante programación propia realizada en Python, además, resolveremos un problema clásico del ML, como es el reconocimiento de números escritos a mano usando el algoritmo AML, con la finalidad de poder compararlo con su resolución a través de algoritmos ya conocidos de ML/ Deep Learning, y obtener así resultados empíricos comparativos sobre la eficiencia de ambos.

Palabras clave: *Algebraic Machine Learning – Algoritmo de Clasificación – Átomos – Deep Learning – Grafos – Identificación de barras verticales – Machine learning – Overfitting – Semirretículos.*

Abstract

Machine Learning (ML) is an important branch of Artificial Intelligence, developed in recent times, that allows us to learn characteristics and predict the value of variables for which we only have examples. However, several of the classical ML methods are usually based on function minimization to fit a large number of parameters in a randomly created model. This method presents several problems, such as the difficulty in finding minima of complicated functions with many variables, or that sometimes the so-called "overfitting" appears, i.e. the model fits too closely to the training data and does not generalize well.

In this paper we will present the alternative proposed in [10] for this type of algorithms, based on algebraic structures, mainly graphs and semigroups, to find a predictive model. This new approach, called Algebraic Machine Learning (AML), is completely parameter-free, and furthermore, when applied in experiments, no evidence of overfitting has been found.

Although it can be applied to more types of problems, in this work we are going to study its performance as a classification algorithm, that is, oriented to obtain a model that allows us to predict whether an element possesses a certain property, starting from a set of elements of which we know a priori if it fulfills this property.
We will explain how to find an algebraic representation of the elements to be classified, as well as of the property studied, with respect to a special set of elements that we will call "atoms", to then see step by step the AML algorithm and how it is applied to a very simple problem, such as the identification of vertical black bars in 2x2 grids of white or black pixels.

Using our own programming in Python, we will also solve a classic ML problem, such as the recognition of handwritten numbers using the AML algorithm, in order to be able to compare it with its resolution by means of already known ML/Deep Learning algorithms, in order to obtain comparative empirical results on the efficiency of both.

Keywords: *Algebraic Machine Learning – Atoms – Classification Algorithm – Deep Learning – Graphs – Identification of vertical barss – Machine learning – Overfitting – Semilattices.*

Contenido

Agradecimientos III

Resumen/Abstract V

Introducción IX

1. Antecedentes 1
1.1. Machine Learning *(enfoque estadístico)* 1
1.2. AML: enfoque algebraico del ML 5

2. Construcción de las estructuras algebraicas del algoritmo AML 7
2.1. Problema de identificación de la barra vertical 7
2.2. Codificación de los datos iniciales 8
2.3. Datos de entrenamiento del modelo AML 9
2.4. Grafo G del problema AML 10
2.5. Semirretículo M del problema AML 12
2.6. Grafo invertido G^* del problema AML 13
2.7. Semirretículo M^* del problema AML 13
2.8. Estructura algebraica completa del algoritmo AML 14
2.9. Cómo realizar predicciones (avance) 16

3. Algoritmo AML 17
3.1. Inicio del algoritmo: relaciones de entrenamiento en G^* 18
3.2. Restricciones de traza 18
3.3. Obtención de restricciones negativas de traza y relaciones negativas 19
3.4. Obtención de restricciones positivas de traza 22
3.5. Sparse crossing: obtención de relaciones positivas 25
3.6. Reducción del número de átomos de G y G^* 31
3.7. Uso el Modelo AML y evaluación de su calidad 33

3.8. Preparación para aprendizaje por lotes: términos y relaciones de fijación 35
3.9. Aprendizaje por lotes: ciclos de entrenamiento 36

4. Resumen: Algoritmo AML paso a paso 39
4.1. Inicialización del Algoritmo AML 39
4.2. Ciclo n-ésimo del aprendizaje por lotes 41
4.3. Predicción a través del Modelo AML 42

5. A menos átomos, menor error 43
5.1. Cálculo de la tasa de error del algoritmo AML 43

6. Conclusiones y líneas futuras 47
6.1. Conclusiones 47
6.2. Líneas futuras 48

Bibliografía 49

Poster 51

Introducción

El Machine Learning (ML) es una importante rama dentro de la Inteligencia Artificial, desarrollada en los últimos tiempos, que nos permite aprender características y poder predecir el valor de variables para las cuales solo tenemos ejemplos. Sin embargo, varios de los métodos clásicos de ML suelen basarse en la minimización de funciones para ajustar una gran cantidad de parámetros en un modelo creado aleatoriamente. Este método presenta varios problemas, como puede ser la dificultad en encontrar mínimos de funciones complicadas y con muchas variables, o que en ocasiones aparece el denominado "overfitting", es decir, el modelo se ajusta demasiado a los datos de entrenamiento y no generaliza bien.

En este trabajo explicamos una alternativa a este tipo de algoritmos, publicada en [10], basada en construcciones algebraicas, que utiliza grafos y semirretículos para encontrar un modelo predictivo, dando lugar al denominado *Algebraic Machine Learning (AML)*. Este algoritmo es completamente libre de parámetros, y además, al aplicarlo en experimentos no se ha encontrado evidencia de *overfitting*.

Aunque puede aplicarse a más tipos de problemas, aquí estudiaremos el funcionamiento del AML como algoritmo de clasificación, es decir, orientado a obtener un modelo que nos permita predecir si un elemento posee una cierta propiedad, partiendo iniciamente de un conjunto de elementos de los que sabemos a priori si cumple dicha propiedad. Explicaremos cómo encontrar una representación algebraica de los elementos a clasificar, así como de la propiedad estudiada, respecto de un conjunto especial de elementos que llamaremos "átomos".

En cuanto a la estructura del trabajo, comenzamos el capítulo 1 con un repaso sobre el ML y los diferentes tipos de algoritmos existentes, para contextualizar la rama que estudiaremos. Recordamos además los algoritmos más usados actualmente por las empresas, y la finalidad de los mismos.

En el capítulo 2, explicamos la construcción de las estructuras algebraicas necesarias para obtener el Modelo AML, que incluyen dos grafos y dos semirretículos asociados. Además, tal y como se hace en [10], formulamos un ejemplo sencillo, el de identificación de barras negras verticales en cuadrículas 2x2

de píxeles blancos o negros, creando las correspondientes estructuras algebraicas para dicho problema.

En el capítulo 3, mostramos cómo transformar las estructuras algebraicas construidas en el capítulo anterior, explicando paso a paso los procedimientos necesarios para obtener el modelo, y aplicándolos a suz vez al problema de la barra vertical. También detallamos la forma de realizar las predicciones una vez obtenido el modelo AML, cómo medir la calidad del modelo y cómo mejorarlo mediante el *aprendizaje por lotes*.

En el capítulo 4 presentamos un esquema resumido de todo el proceso de creación del modelo AML. Este esquema ayudará al/la lector/a interesado/a en implementarlo, proporcionando una guía clara sobre qué procedimientos usar y en qué orden se deben usar dentro del proceso general del algoritmo AML.

El capítulo 5 se centrará exclusivamente en cómo se calcula una cota superior del error esperado en las predicciones del modelo. Además, se explicará la demostración mostrada en [10] que prueba que esta tasa de error decrece con el cardinal de los átomos del modelo, lo que significa que cuantos menos átomos, menos tasa de error.

Por último, en el capítulo 6 comentamos algunas conclusiones y destacamos posibles líneas de investigación que podrían estudiarse para continuar el presente trabajo.

Es de destacar que eL AML es una línea reciente de investigación reciente, en la que aún no existen demasiados artículos y trabajos. El presente trabajo ha analizado en profundidad el principal artículo de investigación publicado en este sentido (véase [10]), junto con otros artículos relacionados de los mismos autores ([9], [13]) y la consulta de algunas de las principales referencias de Teoría de Grafos, así como de Machine Learning (enfoque estadístico).

En el presente trabajo hemos re-escrito el contenido estudiado en [10], ajustándonos al estilo matemático "clásico" a través del uso explícito de *definiciones*, *teoremas, proposiciciones y lemas*, junto con sus respectivas demostraciones.

Aparte de lo anterior, destacamos como contribución propia, entre otras, el esquema-resumen del Capítulo Cuarto, algunas propuestas de otpimización del algoritmo AML explicadas en el Capítulo Sexto, así como la programación propia realizada en Python del algoritmo AML que hemos aplicado al problema clásico del ML de reconocimiento de números escritos a mano usando el algoritmo AML. Esto nos permitirá compararlo con su resolución a través de algoritmos ya conocidos de Deep Learning, por ejemplo, basados en redes neuronales, y obtener así resultados empíricos comparativos sobre la eficiencia de ambos.

1

Antecedentes

En este capítulo recordaremos brevemente el concepto de Machine Learning (ML) y los principales algoritmos que existen. Nos referiremos a éste como *Machine Learning clásico*, o de *enfoque estadístico*, para diferenciarlo de la reciente línea de investigación *Machine Learning Algebraico* (AML), noción que introduciremos al final del presente capítulo, donde destacaremos, además, las ventajas y desventajas de ambos enfoques.

1.1. Machine Learning *(enfoque estadístico)*

Durante décadas se ha intentado comprender cómo piensa el ser humano, es decir, entender cómo percibe, razona, predice y manipula. La *Inteligencia Artificial* va aún más allá, ya que no pretende sólo entender, sino que se esfuerza por intentar construir *entes inteligentes*.

El término "Inteligencia Artificial" fue acuñado por John McCarthy, un científico de la computación estadounidense, en la Conferencia de Dartmouth sobre Inteligencia Artificial en 1956. McCarthy y otros investigadores utilizaron este término para describir la capacidad de las máquinas para realizar tareas que requieren inteligencia humana, como el aprendizaje, la resolución de problemas y la toma de decisiones. Desde entonces, el término Inteligencia Artificial se ha convertido en un campo de estudio y una industria en rápido crecimiento. El objetivo principal de esta ciencia es construir máquinas que puedan realizar tareas valoradas como exigencias de la inteligencia humana, desarrollando sistemas con la capacidad de guardar o almacenar información, realizar cálculos y anticiparse a ciertas tareas. Sus aplicaciones van desde lo más simple, como juegos, traducción de idiomas, o asesorías basadas en la predicción, hasta sistemas más complejos, como robótica, vehículos autónomos o el reciente *ChatGPT*, chatbot de Inteligencia Artificial desarrollado en 2022 por OpenAI basado en un gran modelo de lenguaje entrenado para mantener conversaciones con el usuario y proporcionar información de cualquier tema.

El presente trabajo se centra en una subcategoría de la Inteligencia Artificial, denominada *Machine Learning* (o en español, *Aprendizaje Automático*), acuñado también por Arthur Samuel en 1959, quien lo definió como *el campo de estudio que le da a las computadoras la habilidad de aprender sin ser explícitamente programadas*.

El Machine Learning se enfoca por tanto en enseñar a las máquinas a aprender y mejorar de manera autónoma a través de la experiencia, o más concretamente, en diseñar algoritmos (modelos) que pueden realizar tareas específicas, a través del análisis masivo de datos y la Estadística, permitiendo así el reconocimiento de patrones complejos e inferencias para comprender datos pasados que pueden resolver problemas en el futuro.

Tom Mitchell (1998) caracteriza un problema de Machine Learning bien planteado de la siguiente manera: *Un programa se dice que aprende de la experiencia E respecto a alguna tarea T y alguna medida de eficacia P si su eficacia en T, medida por P, mejora con la experiencia E.*

La principal diferencia entre la programación clásica y el ML radica en que la primera parte de datos iniciales y unas reglas (algoritmo) para tener las respuestas, mientras que en ML se parte de datos iniciales y de la respuesta deseada, obteniéndose las reglas (modelo entrenado). A partir de este último, aplicándolo a nuevos datos, se obtienen las respuestas buscadas.

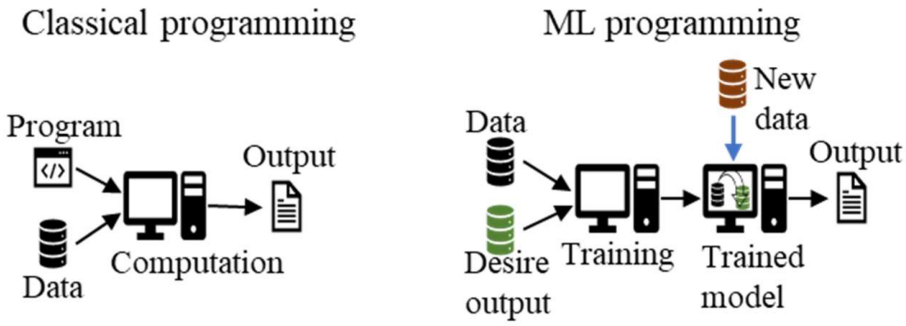

Figura 1.1
Programación clásica versus programación en ML.
https://encyclopedia.pub/entry/27451

El ML ha evolucionado enormemente en los últimos años gracias al aumento en la cantidad de datos disponibles, el desarrollo de técnicas más avanzadas y el incremento en la capacidad computacional de las máquinas.

Por su carácter transversal, presenta un gran atractivo para ser empleado casi en cualquier tipo de empresa y en cualquier tipo de actividad, a partir de los conjuntos de datos de interés generados por las mismas, que son empleados para ser procesados por los algoritmos de ML y posteriormente, para extraer

patrones, y de aquí, conclusiones en base a la predicción. Algunas de las aplicaciones más usadas son: predicción de fraude en las reservas, etiquetado de fotos, predicción de fallos de una máquina, predicción de stock en un almacén, sistemas de recomendación, filtros de *Spam* en los correos, conducción automática, etc.

Hay varias categorías dentro del ML en función de la naturaleza de la retroalimentación del que disponga el sistema de aprendizaje [12]. Cada categoría tiene su aplicación a determinados ámbitos.

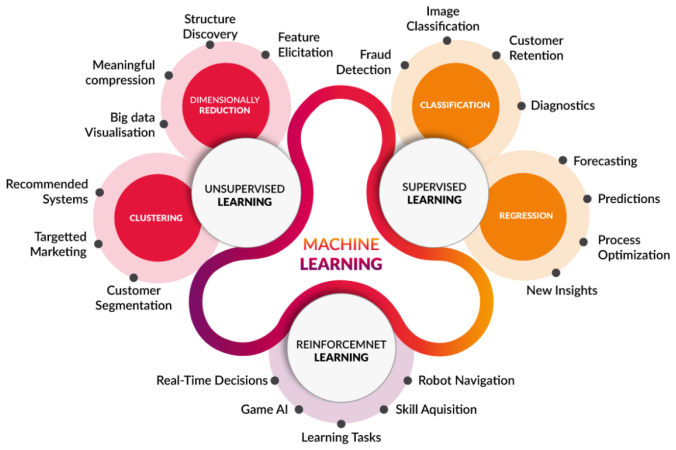

Figura 1.2
Aplicaciones del ML. https://otech.uaeh.edu.mx/noti/index.php/machine-learning/los-conceptos-de-machine-learning-y-deep-learning-en-la-industria/

Tipos de algoritmos de ML:

- **Aprendizaje Supervisado**.
  Para la creación del modelo predictivo se trabaja con datos (entradas) y sus salidas deseadas, con el objetivo de aprender una regla general que relacione las entradas con las salidas. Conocer dichos valores califica al conjunto de datos como "etiquetado" y permite que el algoritmo descifre los patrones existentes entre los datos, creando un modelo capaz de reproducir las mismas reglas subyacentes con la nueva información, es decir, aplicando el modelo a nuevos datos de entrada, podemos predecir la salida.
  Existen dos tipos de Aprendizaje Supervisado, dependiendo de si la variable a predecir es cuantitativa o cualitativa: de Regresión o de Clasificación, respectivamente. En los modelos de Regresión, las salidas son valores reales, por ejemplo, la predicción del número de prendas que se venderán un determinado día, o el precio de mercado de un coche usado. Entre los algoritmos de regresión más conocidos se encuentran: *Regresión Lineal* y *Regresión Polinómica*.

En los modelos de Clasificación, los datos tienen como salida una etiqueta discreta, por ejemplo, la clasificación de imágenes de ropa o el análisis de sentimientos en las redes sociales entre un conjunto finito de etiquetas posibles. Los algoritmos más usados son: *Naive Bayes*, *Árboles de desición*, *Regresión Logística*, *K-Vecinos* o *Support Vector Machine*, entre otros.

- **Aprendizaje No Supervisado**. En la creación del modelo predictivo, no se proporcionan etiquetas al algoritmo de aprendizaje, dejándolo que encuentre por sí mismo patrones escondidos en los datos, incluso aún sin saber de que existen tales patrones.
  Existen tres tipos de modelos de Aprendizaje No Supervisado: Agrupamiento (o Clustering), Reducción de dimensionalidad y Reglas de asociación. Los algoritmos de Clustering definen una métrica de similitud o distancia que les sirve para comparar datos y agruparlos. Se usan, por ejemplo, para agrupar clientes con comportamientos similares, a los que digirigr acciones comerciales, o identificar acciones sospechosas de fraude. Los algoritmos más populares de Clustering son: *K-means*, *Mean-Shift* o *DBSCAN*.
  La Reducción de Dimensionalidad se utiliza para reducir el tiempo de entrenamiento de los modelos, mejorar su rendimiento o agilizar la representación visual de los datos. Entre los algoritmos más conocidos se encuentran: *Principal Component Analysis (PCA)*, *Singular Value Decomposition(SVD)*, *Latent Dirichlet allocation (LDA)*, *Latent Semantic Analysis (LSA, pLSA, GLSA)* o *t-SNE (para visualización)*.
  El aprendizaje por Reglas de Asociación se utiliza para descubrir relaciones entre variables en grandes conjuntos de datos, por ejemplo, de los datos de ventas de un supermercado se podría deducir que un consumidor que compra cebollas y verdura a la vez, es probable que compre también carne. Estos modelos son útiles para tomar decisiones sobre marketing, como precios promocionales para ciertos productos, dónde ubicar estos dentro del supermercado, o en otras muchas áreas como el *web mining*, la detección de intrusos o la bioinformática. Los algoritmos más usados son: *Apriori*, *Euclat* y *FP-growth*.

- **Aprendizaje por refuerzo**.
  El Aprendizaje por Refuerzo tiene su origen en la psicología del aprendizaje animal y en la teoría del *aprendizaje de prueba y error*. Se aplica a problemas no supervisados que reciben re-alimentaciones o refuerzos (gana o pierde), y se basa precisamente en recompensar los comportamientos deseados y penalizar los no deseados. Un agente es capaz de aprender a través de prueba y error en un ambiente dinámico e incierto, y mapear situaciones de acciones para maximizar una cierta función de recompensa.
  Ejemplos de este tipo de aprendizaje son los modelos de juegos enfrentándose a un oponente [2], como el ajedrez o AlphaGo, o la aplicación a los vehículos

autónomos. Los algoritmos de Aprendizaje por Refuerzo más conocidos son: *Q-Learning*, *SARSA*, *DQN*, *A3C* y *Genetic algorithm*.

- **Aprendizaje Profundo (Deep Learning). Redes Neuronales.** El Aprendizaje Profundo, basado en redes neuronales complejas, está diseñado para emular el funcionamiento del cerebro humano, de forma que el ordenador "se capacita"para lidiar con abstracciones a partir de la implementación de técnicas de *Big Data*, y es capaz de buscar sugerencias en relación al tema planteado. Estos algoritmos funcionan mediante una serie de "capas"de neuronas que realizan cálculos mediante el *input* recibido de la anterior capa hasta llegar a un resultado final. Se usan en aplicaciones de reconocimiento de imagen, voz y visión. Las arquitecturas más conocidas son: *Perceptrón*, *Convolutional Network (CNN)*, *Recurrent Networks (RNN)* o *Autoencoders*.

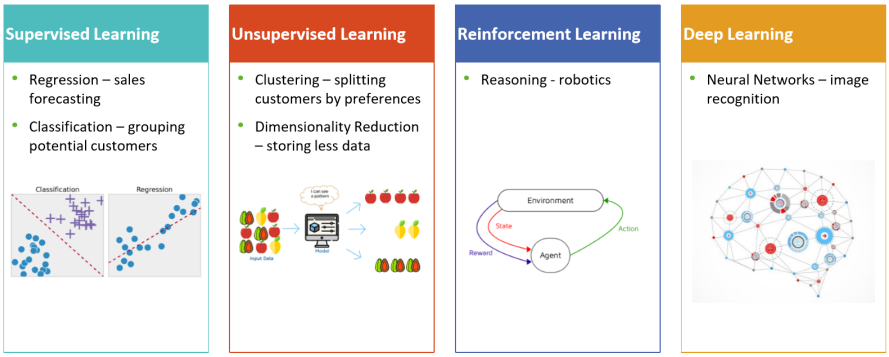

Figura 1.3
Clasificación del ML (Estadístico).
https://es.clariba.com/machine-learning-for-business

1.2. AML: enfoque algebraico del ML

Los algoritmos de ML anteriores, de enfoque estadístico, utilizan usualmente métodos de minimización de ciertas funciones de error para ajustar muchos parámetros con procedimientos heurísticos, perdiendo en ocasiones capacidad de generalización a otros conjuntos de datos (pérdida de generalidad del modelo). Estas funciones, además, suelen tener geometrías complejas, y navegar por sus superficies a menudo requieren grandes conjuntos de datos y métodos especiales para evitar que el método se atasque en los mínimos locales.

El AML ha aparecido recientemente como una alternativa al ML. Se basa en un nuevo enfoque matemático en el que el problema a resolver se embebe

en determinadas estructucturas alebraicas, que combinan símbolos y relaciones definidas por los datos inciales (datos de entrenamiento) con símbolos autogenerados por el propio método, que captan de forma natural las principales características de los datos, permitiendo así el aprendizaje.

Concretamente, el problema se embebe en semirretículos y grafos. Tras la codificación de los datos de entrenamiento a estas estructuras, el aprendizaje tiene lugar mediante transformaciones algebraicas sobre las mismas (nodos y arcos de los grafos), usando, entre otros, un método denominado *Sparse Crossing*, y minimizando a su vez el número de cierto tipo de nodos que se van autogenerando, los *átomos*. El modelo AML final viene caracterizado por el conjunto de átomos de los grafos transformados en el último paso del algoritmo, o lo que es lo mismo, la composición final de uno de los semirretículos.

El enfoque algebraico presenta varias ventajas respecto al ML clásico, por ejemplo, no realiza minimización de funciones, sino una reducción del número de átomos del grafo, y por tanto, no tiene el riesgo de quedarse atrapado en mínimos locales. Además, es un algoritmo estocástico y discreto, sin operaciones de coma flotante, y no necesita grandes cantidades de datos de entrenamiento. En [10] los autores justifican que cuanto más pequeño es el conjunto de átomos, mayor es la precisión del modelo. Los autores también argumentan que no se han encontrado indicios de *overfitting*.

El AML es puramente simbólico, no utiliza parámetros y no se basa en el ajuste o la minimización de errores, y por tanto, no se pierde generalidad en los modelos, abriéndose así una vía alternativa hacia la Inteligencia Artificial.

Clasical Machine learning	**Algebraic Machine learning**
Function minimization	Cardinal minimization
Error functions tend to be very complex and it is hard to find minima or to not get stuck on local minima	Does not need to find minima
Depends on a huge number of parameteres which might need tuning with heuristic procedures.	Parameter free
There is risk of overfitting	No evidence of overfitting has been found

Figura 1.4
Diferencias entre ML Estadístico y el Algebraico.

Otras diferencias entre el AML y los métodos basados en enfoque estadístico de Inteligencia Artificial Simbólica y Redes Neuronales puede encontrarse en [9].

2

Construcción de las estructuras algebraicas del algoritmo AML

Aunque podría aplicarse a otros tipos de problemas, en este trabajo nos centraremos en cómo se crea un modelo predictivo de AML a partir de unos datos iniciales correspondientes a un problema Supervisado de Clasificación (véase el apartado 1.1).

Dado un conjunto de elementos (datos), Ω, buscamos obtener un modelo $\mathcal{M}$ tal que $\forall x \in \Omega$, $\mathcal{M}$ sea capaz de determinar si x posee una cierta propiedad P o no. Como se adelantó al final del Capítulo 1, el algoritmo de AML requiere inicialmente embeber dichos datos Ω en determinadas estructuras algebraicas, donde se aplicarán posteriormente las transformaciones propias del método, que serán explicadas en el capítulo 3.

En el presente capítulo se detalla cómo llevar a cabo la codificación de los datos iniciales a través de dos estructuras algraicas determinadas -semirretículos y grafos-, que captarán de forma natural las principales características de los datos de entrada y permitirán el aprendizaje del modelo AML. Para ilustrar mejor este proceso, iremos aplicando cada paso al problema de identificación de la barra vertical, enunciado a continuación.

2.1. Problema de identificación de la barra vertical

Para ilustrar el funcionamiento del algoritmo de AML, en este trabajo nos centraremos en un problema sencillo, el problema de *identificación de la barra vertical*:

Sea Ω el conjunto de las cuadrículas de tamaño $2x2$ (4 píxeles) donde cada píxel puede tomar el color negro o blanco. El problema consiste en construir un modelo $\mathcal{M}$ tal que $\forall T \in \Omega$, $\mathcal{M}$ es capaz de predecir si T tiene una barra vertical negra (un píxel negro sobre otro), o no.

A los elementos T de Ω los llamaremos *términos*. Representaremos por $v \leq T$ la existencia de una barra vertical negra en T y por $v \not\leq T$ la inestistencia de la misma. Más adelante se verá la justificación de esta notación (donde $\leq$

será el orden definido en la fórmula (2.4)).

En la siguiente figura podemos ver dos términos de Ω, uno con barra vertical negra y el otro sin barra.

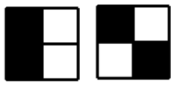

Figura 2.1
Ejemplos de $v \leq T$ y $v \not\leq T$, respectivamente.

2.2. Codificación de los datos iniciales

El primer paso para resolver el problema planteado en 2.1 es codificarlo usando un conjunto de *constantes*, es decir, se deben buscar unos elementos fijos mediante los que podamos representar cualquier término de Ω. En nuestro caso, se deben buscar un conjunto fijo de elementos (constantes) con las que se puedan formar cualquier cuadrícula de $2x2$ píxeles.

Las constantes que se fijan para el problema de identificación de la barra vertical son los 8 píxeles blancos y negros, junto con su posición en la cuadrícula $2x2$ (véase Figura 2.2), que denotamos por $c_1, \ldots, c_8$, respectivamente.

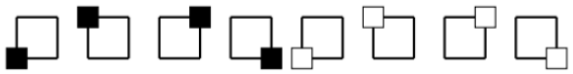

Figura 2.2
Constantes del problema de identificación de la barra vertical.

Es claro que:

$$\forall T \in \Omega, \exists i_1, \ldots, i_4 : T = c_{i_1} \odot \cdots \odot c_{i_4}, i_j \in \{1, \ldots, 8\},$$

donde, como veremos en el apartado 2.5, $\odot$ será la operación definida en el semirretículo donde embeberemos el problema, y que se comporta como una "unión de constantes". La figura 2.3 nos muestra un ejemplo de representación de un término como unión de constantes.

Figura 2.3
Respresentación de un término a través de las constantes.

2.3. Datos de entrenamiento del modelo AML

Al igual que cualquier otro algoritmo de ML Supervisado, necesitamos partir de unos datos de entrenamiento con los que se puedan construir el modelo de AML.

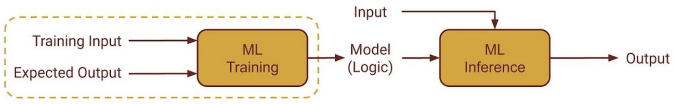

Figura 2.4
Creación del modelo ML a partir de los datos.

El conjunto de datos de entrenamiento será una muestra D de términos de Ω ($D \subseteq \Omega$), y, en nuestro caso, como se explicó en el apartado 2.2, para todo término $T \in D$, o bien $v \leq T$ (T contiene una barra negra vertical), o bien $v \not\leq T$ (T no contiene una barra negra vertical).

En el ejemplo que usaremos en este trabajo para ilustrar cada paso del algoritmo AML, tomaremos D como el conjunto de términos indicados en la Figura 2.5.

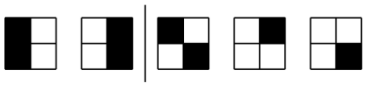

Figura 2.5
Subconjunto fijo de términos con el que construiremos el modelo AML.

En primer lugar, separaremos el conjunto D en dos subconjuntos

$$D = D^+ \cup D^-,$$

donde

$$D^+ = \{T^+ \in D / v \leq T^+\}, \quad D^- = \{T^- \in D / v \not\leq T^-\}.$$

Esta primera clasificación de los datos de entrenamiento es en base a la etiqueta de salida que se conoce inicialmente en cualquier problema de ML Supervisado (véase 1.1).

En nuestro caso de estudio, $D = \{T_1^+, T_2^+, T_1^-, T_2^-, T_3^-\}$, cuya representación visual se corresponde con los elementos mostrados respectivamente en la Figura 2.5.

A partir del conjunto D etiquetado adecuadamente, constuimos el conjunto de *relaciones de entrenamiento*, que será el input del algoritmo AML, como:

$$R = R^+ \cup R^-,$$

donde

$$R^+ := \{v \leq T_i^+ / T_i^+ \in D^+\} \quad R^- := \{v \not\leq T_i^- / T_i^- \in D^-\}, \tag{2.1}$$

es decir, $R = \{v \leq T_1^+, v \leq T_2^+, v \not\leq T_1^-, v \not\leq T_2^-, v \not\leq T_3-\}$. El objetivo a conseguir en el algoritmo AML es que las relaciones anteriores se cumplan para el orden $\leq$ que será definido en la fórmula (2.4).

A partir de las relaciones de entrenamiento (2.1), construiremos a continuación las estructuras algebraicas en las que se embebe nuestro problema AML.

2.4. Grafo G del problema AML

En el contexto anterior, definimos a continuación el grafo dirigido $G = (V, E)$ del problema de AML.

Definición 2.1. *Sea $G = (V, E)$ el grafo determinado por el conjunto de nodos (o vértices):*

$$V = D \cup C(G) \cup A(G), \text{ donde:}$$

- ***Términos:*** *$D = \{T_1^+, T_2^+, T_1^-, T_2^-, T_3^-\}$, cada uno de ellos formados como "unión"de constantes (véase apartado 2.2),*
- ***Constantes:*** *$C(G)$, consistente en las constantes usadas para codificar el problema (véase Figura 2.2) junto con una constante adicional, v, introducida en los apartados anteriores, que representa la propiedad buscada, esto es, $C(G) = \{c_1, \ldots, c_8, v\}$,*
- ***Átomos:*** *$A(G)$, que serán unos nuevos elementos que iremos añadiendo durante el proceso de construcción del modelo e inicialmente sólo tenemos un átomo especial, que que denotaremos por 0, esto es, $A(G) = \{0\}$,*

y usando la notación $a \to b$ para referirnos al arco del nodo a al nodo b, el conjunto de arcos E se construye partiendo de los siguientes axiomas:

1. *Sea $T \in D$, $\{c_1, \ldots, c_n\} \subseteq C(G) : T = \bigodot_{i=1}^n c_i \implies c_i \to T$, $1 \leq i \leq n$;*
2. *Sean $T, S \in D$ tal que $T = \bigodot_{i=1}^n c_i$ y $S = \bigodot_{j=1}^m c'_j$, $\{c'_1, \ldots, c'_m\} \subseteq \{c_1, \ldots, c_n\} \implies T \to S$;*

3. $\forall a \in D \cup C(G),\ 0 \to a$;
4. $\forall a \in V,\ a \to a$; *(Propiedad Reflexiva)*
5. $\forall\ a, b, c \in V : a \to b\ \wedge\ b \to c \implies a \to c$ *(Propiedad Transitiva).*

En esta construcción inicial del grafo G y en las sucesivas transformaciones que sufrirá en el algoritmo AML (capítulo 3), se mantendrán esas condiciones, y además que $\forall a, b \in V$ tal que $a \to b$ y $b \to a$, entonces $a = b$ (Propiedad Antisimétrica), es decir, los arcos son unidireccionales. De aquí, obsérvese que $G = (V, E)$ es un digrafo acíclico transitivamente cerrado (véase [5], [6]).

Definición 2.2. *Sea $D \subseteq \Omega$ el conjunto de entrenamiento de un problema de identificación de la barra vertical y $G = (V, E)$ su grafo asociado. Para todo $x \in V$,*

$$GL(x) = \{b \in V / b \to x\}\ \textit{Nodos predecesores de } x;$$
$$GU(x) = \{b \in V / x \to b\}\ \textit{Nodos sucesores de } x;$$
$$GL^c(x) = GL(x) \cap C(G)\ \textit{Constantes predecesoras de } x;$$
$$GU^c(x) = GU(x) \cap C(G)\ \textit{Constantes sucesoras de } x;$$
$$GL^a(x) = GL(x) \cap A(G)\ \textit{Átomos predecesores de } x;$$
$$GU^a(x) = GU(x) \cap A(G)\ \textit{Átomos sucesores de } x.$$

A continuación se representa el grafo $G = (V, E)$ correspondiente a los datos de entrenamiento fijados en el apartado 2.3. Para hacer más clara la representación, en este trabajo no dibujaremos la dirección de los arcos (siempre será "de abajo hacia arriba"), ni tampoco los arcos implícitos (si $a \to b$ y $b \to c$ no dibujamos el arco $a \to c$ al dedudirse por transitividad, ni tampoco $a \to a$). A esta representación de G se le denomina *Diagrama de Hasse* en Teoría de Grafos.

Este grafo será modificado en diferentes pasos del algoritmo AML, que veremos en el capítulo 3.

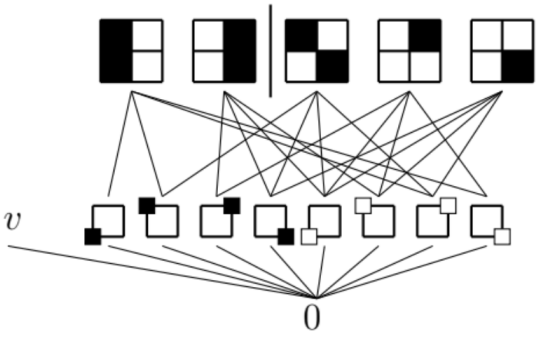

Figura 2.6
Grafo G inicial del algoritmo de AML.

2.5. Semirretículo M del problema AML

A continuación construiremos un semirretículo partiendo del grafo G, pero recordemos previamente la definición de esta estructura algebraica[3]:

Definición 2.3. *Sea M un conjunto y sea $\odot : M \times M \to M$ una aplicación. Diremos que el par $(M, \odot)$ es un semirretículo si $\forall x, y, z \in M$ se cumplen las siguientes propiedades*

1. *Asociatividad: $x \odot (y \odot z) = (x \odot y) \odot z$;*
2. *Conmutatividad: $x \odot y = y \odot x$;*
3. *Idempotencia: $x \odot x = x$.*

De la definición, se deduce fácilmente la siguiente propiedad.

Lema 2.4. *Si $(M, \odot)$ es un semirretículo, $\odot$ induce la siguiente relación de orden (parcial): $\forall x, y \in M$,*

$$x \leq_M y \iff x \odot y = y$$

Definición 2.5. *Sea la aplicación*

$$GL^a : C(G) \cup D \to P(A(G))$$
$$v \to GL^a(v) = \{\phi \in A(G)/\phi \to v\} \text{ (véase definición 2.2)}$$

donde $P(A(G))$ es el conjunto "partes de los átomos de G", esto es, $P(A(G)) = \{B/B \subseteq A(G)\}$. Definimos

$$M := \{GL^a(X)/X \subseteq C(G) \cup D\},$$

es decir, M es el conjunto formado por imágenes de subconjuntos de las constantes y términos de G a través de la aplicación GL^a, sobre el que construye la operación

$$\odot : M \times M \to M$$
$$(x, y) \to x \cup y$$

La operación $\odot$ es claramente conmutativa, asociativa e idempotente, y por tanto $(M, \odot)$ es un semirretículo, que induce el siguiente orden: $\forall A, B \in M$,

$$A \leq_M B \iff A \cup B = B \iff A \subseteq B. \tag{2.2}$$

El objetivo del algoritmo AML será "transformar" el semirretículo M (más bien, del grafo G) hasta que cumpla las relaciones de entrenamiento R, o lo que es lo mismo, que $v \leq_M T^+$ y $v \not\leq_M T^-$, puesto que si conseguimos esto, tendremos una descripción de cada constante en átomos (incluyendo v) que podremos usar para comprobar si un término cualquiera posee dicha propiedad. Esto se explicará más en profundidad al terminar la construcción del algoritmo AML.

No es suficiente con las estructuras que hemos construido hasta ahora, el algoritmo AML necesita una estructura auxiliar.

2.6. Grafo invertido G^* del problema AML

Vamos a crear otro nuevo grafo que denotaremos por $G^* = (V^*, E^*)$, que nos va a ayudar a conseguir las relaciones buscadas en el grafo G. En [10] denominan "dual" a G^* (y a su semirretículo asociado), sin embargo, para evitar confusión con definiciones establecidas en teoría de grafos que no representan el mismo concepto, en este trabajo usaremos el concepto de *Grafo invertido*, ya que básicamente su contrucción se basa en invertir los arcos de G.

Por cada nodo de G, "crearemos" un nodo de G^*. Usaremos la notación $[a]$ para referirnos al nodo de G^* correspondiente al nodo a de G.

Definición 2.6. *El grafo $G^* = (V^*, E^*)$ viene determinado por el conjunto de nodos (o vértices):*

$$V^* = A^* \cup C(G^*) \cup A(G^*), \text{ donde:}$$

- ***Imágenes de átomos:*** $A^* = \{[\phi]/\phi \in A(G)\}$, *(inicialmente,* $A^* = \{[0]\}$*)*
- ***Constantes:*** $C(G^*) = \{[a]/a \in C(G)\} \cup \{[T_i]/T_i \in D\}$, *es decir, las imágenes mediante* $[\_]$ *de las constantes* $C(G)$ *y los términos de* G,
- ***Átomos:*** $A(G^*)$, *que serán añadidos en los distintos pasos del algoritmo, y que incialmente tendrá un nodo especial que denotaremos* 0^*, *esto es,* $A(G^*) = \{0^*\}$,

y el conjunto de arcos, E^*, que se construirá a partir de los siguientes axiomas

1. $\forall\ a, b \in V : a \to b \implies [b] \to [a]$ *en* V^*;
2. $\forall a \in C(G^*) \cup A^*,\ 0^* \to a$;
3. $\forall a \in V^*,\ a \to a$ *(Propiedad Reflexiva);*
4. $\forall\ a, b, c \in V^* : a \to b \ \wedge\ b \to c \implies a \to c$ *(Propiedad Transitiva);*

$G^* = (V^*, E^*)$ también es un digrafo acíclico transitivamente cerrado, ya que las propiedades anteriores se mantendrán en las sucesivas transformaciones realizadas en el algoritmo AML, además, que $\forall a, b \in V^*$ tal que $a \to b \ \wedge\ b \to a \implies a = b$ (Propiedad Antisimétrica). Al igual que con G, usaremos el Diagrama de Hasse de $G^* = (V^*, E^*)$ (ver Figura 2.7), en el que la dirección de las flechas también se leen "de abajo a arriba".

2.7. Semirretículo M^* del problema AML

De forma análoga al caso de G y M, G^* induce la creación de un semirretículo M^*.

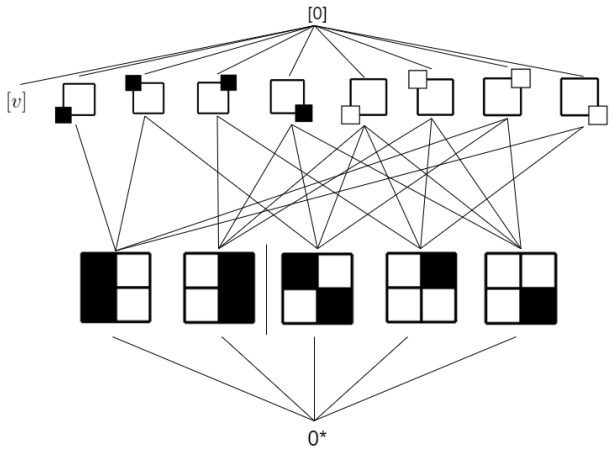

Figura 2.7
Grafo G^* inicial del algoritmo de AML

Definición 2.7. *Mediante la aplicación*

$$GL^a : V^* \to P(A(G^*))$$
$$x \to GL^a(x) = \{\phi \in A(G^*)/\phi \to x\} \text{ (véase definición 2.2)}$$

donde $P(A(G^*))$ es el conjunto "partes de los átomos de G^* y $GL^a(d)$ se refiere a los átomos en G^*.Definimos

$$M^* = \{GL^a(X)/X \subseteq V^*\},$$

es decir, M^* es el conjunto formado por imágenes de subconjuntos de los vétices de G^* a través de la aplicación GL^a.

De forma similar al caso de M, definimos la operación

$$\odot : M^* \times M^* \to M^*$$
$$(a, b) \to a \cup b$$

La operación $\odot$ es conmutativa, asociativa e idempotente, y por tanto $(M^*, \odot)$ es un semirretículo, que induce el siguiente orden: $\forall A, B \in M^*$,

$$A \leq_{M^*} B \iff A \cup B = B \iff A \subseteq B. \tag{2.3}$$

2.8. Estructura algebraica completa del algoritmo AML

Ya construidos los grafos G y G^* y los semirretículos M y M^*, como inicialización del algoritmo AML consideraremos la estructura algebraica completa ([10]).

Definición 2.8. *En G y G^*, tomemos la siguiente relación entre los nodos:*

$$\text{Sean } a, b \in V \cup V^*, \quad a \leq b \iff (\forall \phi \in A(G) \cup A(G^*) : \phi \to a \implies \phi \to b) \tag{2.4}$$

La relación anterior es un orden, con la observación de que sólo los elementos de V pueden compararse entre sí, y sólo los elementos de V^* pueden compararse entre sí. Además, es claro por construcción, que si comparamos elementos de V, los átomos ϕ mencionados estarán solo en $A(G)$, y si comparamos elementos de V^* los átomos mencionados estarán en $A(G^*)$. Asimismo, del orden (2.4) se deduce:

$$\forall a, b \in V \cup V^* : \ a \to b \implies a \leq b, \tag{2.5}$$

ya que, por transitividad, $\phi \to a \wedge a \to b \implies \phi \to b$.

Además, consideremos la unión de los semirretículos $(S, \leq)$ donde $S = M \cup M^*$ y $\leq$ denota al siguiente orden:

$$A \leq B = \begin{cases} A \leq_M B, & A, B \in M \\ A \leq_{M^*} B, & A, B \in M^* \end{cases} \tag{2.6}$$

y donde los elementos de M y M^* no son comparables.

Veamos a continuación que los respectivos órdenes en la estructura de semirretículos $S = M \cup M^*$ y en la grafos $G \cup G^*$ son equivalentes.

Lema 2.9. *Sean $b, c \in V \cup V^*$,*

$$b \leq c \text{ en } G \cup G^* \iff GL^a(b) \leq GL^a(c) \text{ en } S = M \cup M^*,$$

donde el primer orden se refiere al orden (2.4), y el segundo, a (2.6) definido en los semirretículos.

Demostración. Sean $b, c \in V \cup V^*$. Veamos la implicación hacia la derecha. Tenemos que

$$b \leq c \implies \forall \phi \in A(G), \phi \to b \implies \phi \to c,$$

por tanto,

$$\forall \phi : \phi \in GL^a(b) \implies \phi \in GL^a(c)$$
$$\implies GL^a(b) \subseteq GL^a(c) \implies GL^a(b) \leq GL^a(c).$$

Veamos ahora la implicación hacia la izquierda:

$$GL^a(b) \leq GL^a(c) \implies \forall \phi \in GL^a(b), \phi \in GL^a(c) \implies$$
$$(\phi \to b \implies \phi \to c) \implies b \leq c.$$

□

2.9. Cómo realizar predicciones (avance)

Aunque aún no hayamos construido un modelo funcional, y aunque se explique con más detalle en los capítulos posteriores, daremos un avance aquí de cómo se usará la estructura algebraica definida en el apartado 2.8 para realizar predicciones.

Tras varias transformaciones sobre los grafos $G \cup G^*$ y la estructura algebraica $S = M \cup M^*$, la salida del algoritmo AML, el modelo $\mathcal{M}$ de AML, será estas mismas estructuras, pero con muchos más átomos.

Sea $T \in \Omega$ un término (cuadrícula $2x2$) que queramos clasificar, es decir, sobre la que queremos inferir si tiene una barra negra vertical, esto es, si $v \leq T$.

Como previamente hemos codificado el problema, sabemos que $\exists c_1, \ldots, c_n \in C(G)$ tales que

$$T = \bigodot_{i=1}^{n} c_i.$$

Con el modelo $\mathcal{M}$ tenemos una descripción en átomos de cada una de estas constantes y de la constante v, es decir, conocemos los conjuntos:

$$GL^a(v),\ \ GL^a(c_i), \forall i \in \{1, \ldots, n\}.$$

Entonces T tiene una barra negra vertical sí, y solo si, por 2.9,

$$v \leq T \iff GL^a(v) \subseteq GL^a(T).$$

Como

$$GL^a(T) = \bigcup_{i=1}^{n} GL^a(c_i),$$

la predicción se reduce a:

$$v \leq T \iff GL^a(v) \subseteq \bigcup_{i=1}^{n} GL^a(c_i). \tag{2.7}$$

Nótese que la codificación del poblema de identificación de barra vertical y la construcción de las estructuras algebraicas explicadas en el presente capítulo, pueden ser aplicadas a cualquier problema de ML Supervisado de Clasificación para predecir si se cumple una determinada propiedad P, cambiando "T tiene una barra vertical negra"por "T cumple la propiedad P", que, análogamente, será equivalente a que $v \leq T$, donde v es la constante que usamos para codificar dicha propiedad P.

3

Algoritmo AML

Después de haber codificado el problema de identificación de la barra vertical y de haber construido las estructuras algebraicas necesarias, en este capítulo nos centraremos explicar cómo obtener el modelo AML.

Nuestro objetivo es conseguir un modelo $\mathcal{M}$, consistente básicamente en el semirretículo M después de pasar por una serie de transformaciones, de tal forma que tengamos una descripción en átomos $A(G)$ de cada una de las constantes del problema (Figura 2.2) y de la constante v, y donde se cumplan las relaciones R de entrenamiento: $v \leq T^+, \forall T^+ \in D^+ \wedge v \not\leq T^-, \forall T^- \in D^-$ para el orden definido en G (véase fórmula (2.4)). Esto nos permitirá realizar predicciones para cualquier término T (cuadrícula $2x2$) como se describe en la fórmula (2.7).

Este resultado se conseguirá añadiendo átomos en los grafos G y G^* inicialmente construidos, y dado que las estructuras M y M^* se construyen a partir de éstos, los semirretículos también irán evolucionando consecuentemente en cada paso.

Todo el capítulo se basa en el método explicado en [10], no obstante, todas las demostraciones aquí presentadas son propias, algunas como alternativas a las del artículo, y otras relativas a propiedades no recogidas en el mismo.

Nota. A la hora de implementar los algoritmos que se explican en esta sección, siempre que se añada un nuevo átomo y arco $\phi \to a$ en el grafo G, deben añadirse los arcos $[a] \to [\phi]$ y $0^* \to [\phi]$ en G^*, así como calcular la clausura transitiva de ambos grafos y trabajar con éstos. De la misma manera, al añadir un nuevo átomo $\zeta \in A(G^*)$ y arista $\zeta \to b$ en G^* debe calcularse la clausura transitiva de G^* y trabajar con éste. En caso de eliminar un nodo a de G, deben eliminarse todos los arcos relacionados con él, así como el nodo $[a]$ de G^* con sus respectivos arcos.

3.1. Inicio del algoritmo: relaciones de entrenamiento en G^*

El primer paso para lograr que se cumpla R es forzar a que se cumplan las relaciones de R invertidas en G^*, es decir añadiremos los siguientes arcos a G^*, que inducirán desigualdades de acuerdo con la fórmula (2.5):

$$\forall v \leq T^+ \in R^+, \text{ añadimos } [T^+] \to [v] \text{ en } G^* \implies [T^+] \leq [v] \text{ en } G^*; \tag{3.1}$$

$$\forall v \not\leq T^- \in R^-, \text{ añadimos un nuevo átomo } \xi \to [T^-] \text{ en } G^* \implies$$
$$\xi \in GL^a([T^-]) \wedge \xi \notin GL^a([v]) \implies [T^-] \not\leq [v]. \tag{3.2}$$

Nota. Si no es posible forzar todas las relaciones invertidas, es porque éstas son inconsistentes, y por tanto, debemos tomar nuevos datos de entrenamiento.

Ejemplo. A partir del grafo G^* construido inicialmente (véase Figura 2.7), añadimos los arcos $[T_1^+] \to [v]$, y $[T_2^+] \to [v]$ y creamos tres nuevos átomos $\zeta_i, i = 1, 2, 3$ con los arcos, $\zeta_i \to [T_i^-], i = 1, 2, 3$, obteniendo:

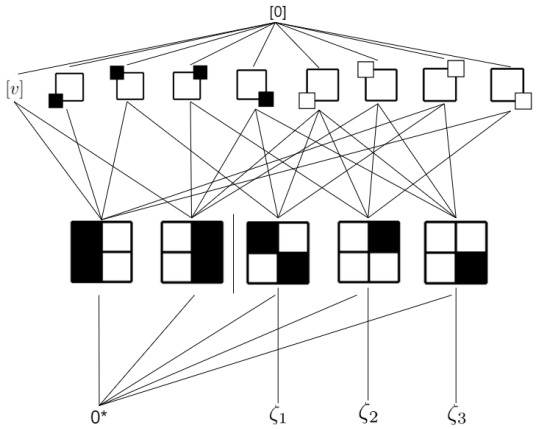

Figura 3.1
Grafo G^* tras forzar relaciones de entrenamiento.

3.2. Restricciones de traza

Definimos a continuación la *traza* de los nodos del grafo G, que posee propiedades que nos serán de utilidad.

Definición 3.1. *Sea $x \in V$, definimos*

$$Tr(x) := \bigcap_{\phi \in A(G):\phi \to x} GL^a([\phi]) \subseteq M^*.$$

Lema 3.2. *Sean* $x, y \in V$, $Tr(y) \not\subseteq Tr(x) \implies x \not\leq y$ *en* G.

Demostración. Sean $x, y \in V$, tal que $x \leq y$, entonces, por fórmula (2.4), tenemos que $\forall \phi \in A(G) : \phi \to x \implies \phi \to y$. De aquí, $Tr(y) = \bigcap_{\phi \to y} GL^a([\phi]) \subseteq \bigcap_{\phi \to x} GL^a([\phi]) = Tr(x)$. □

Corolario. Del lema 3.2 deducimos que las restricciones negativas de traza son condición suficiente para obtener R^- en la estructura de G y M (tomando $x = v$, $y = T_i^-$), y además, por el contrarrecíproco, que las restricciones positivas de traza son una condición necesaria para obtener R^+ (para $x = v$, $y = T_i^+$).

3.3. Obtención de restricciones negativas de traza y relaciones negativas

Del lema anterior deducimos que para obtener $v \not\leq T_i^-$ en G, basta conseguir que $Tr(T_i^-) \not\subseteq Tr(v)$. Para ello, usaremos el Algoritmo 1, aplicable a $(a \not\leq b) \in R^-$, con $a \in C(G)$ y $b \in D^-$ o b es un término de fijación(consultar 3.9):

```
for (a ⋠ b) ∈ R⁻ do
    if Tr(b) ⊆ Tr(a) then
        while c = ∅ do
            c = findStronglyDiscriminantConstant(a, b);
            if c = ∅ then
                Elegimos h ∈ C(G*) : h ∈ GL^c([b])\GL([a]);
                Añadimos un nuevo átomo ζ ∈ G* y el arco ζ → h
            end
        end
        Añadimos un nuevo átomo φ a G y el arco φ → c
    end
end
Function findStronglyDiscriminantConstant(a,b)
    Calcular Ω(a) ≡ {[c]/c ∈ GL^c(a)};
    Inicializar U ≡ Tr(b);
    while U≠ ∅ do
        Elegimos un átomo ζ ∈ U y lo eliminamos de U;
        if Ω(a)\GU(ζ) ≠ ∅ then
            Elegimos [c] ∈ Ω(a)\GU(ζ);
            return c;
        end
    end
    return ∅;
end
```

Algorithm 1: Restricciones negativas de traza.

A continuación vamos a demostrar que tras aplicar Algorithm 1 conseguimos que $\forall\ v \not\leq T_i^- \in R^-$, pero para ello necesitaremos el siguiente lema.

Lema 3.3. *Para todo* $x \in V$, $GL^a([x]) \subseteq Tr(x)$.

Demostración. Sea $x \in V$, y consideremos $GL^a(x) = \{\phi_1, \ldots, \phi_n\}$. Según la definición de traza de un elemento, $Tr(x) = \bigcap_{i=1}^{n} GL^a([\phi_i])$, tenemos que:

$$\forall i \in \{1, \ldots, n\}, \phi_i \to x \text{ en } G \implies [x] \to [\phi_i] \text{ en } G^*$$

$$\implies [x] \leq [\phi_i] \text{ en } G^* \text{ (por 2.5)} \implies GL^a([x]) \subseteq GL^a([\phi_i]) \text{en } M^* \text{ (por 2.9)}.$$

Luego, $GL^a([x]) \subseteq \bigcap_{i=1}^{n} GL^a([\phi_i]) = Tr(x)$. □

Proposición 3.4. *Sean $a \in C(G)$ y $b \in D^-$ tal que $Tr(b) \subseteq Tr(a)$, tras aplicar Algorithm 1 se tiene que $Tr(b) \not\subset Tr(a)$, y por tanto, por Lema 3.2, $a \not\leq b$.*

Demostración. Sean $a \in C(G), b \in D^-$, $Tr(b) \subset Tr(a)$. Queremos probar que después de aplicar Algorithm 1, $Tr(b) \not\subset Tr(a) \implies a \not\leq b$. Estudiemos los 2 casos posibles:

1. *Caso 1*. Supongamos que existe una constante $c \in C(G)$ tal que $c \to a$ y tal que $\exists \zeta \in Tr(b) : \zeta \not\to [c]$, esto es, estamos en la situación de que la función *findStronglyDiscriminantConstant(a,b)* encuentra tal c.
   Siguiendo el algoritmo, se sale del primer *while* y creamos un nuevo átomo ϕ y un nuevo arco $\phi \to c$ en G. Veamos que $Tr(b) \not\subset Tr(a)$, demostrando que el $\zeta \in Tr(b)$ descrito anteriormente, no está en $Tr(a)$.
   Como $c \to a$, entonces, por (2.5) y Lema 3.2, $Tr(a) \subseteq Tr(c)$. Además, como $\phi \to c$, entonces $Tr(c) = \bigcap_{\psi \to c} GL^a([\psi]) \subseteq GL^a([\phi])$, y como ϕ es un nuevo átomo que únicamente está anexado a c, tenemos que $GL^a([\phi]) = GL^a([c])$. De las desigualdades anteriores se concluye que $Tr(a) \subseteq GL^a([c])$.
   Hemos elegido c tal que $\zeta \not\to [c]$. Luego, $\zeta \notin GL^a([c])$, y por tanto, $\zeta \not\subset Tr(a)$.
   A partir de aquí, el algoritmo seguiría en el siguiente paso del *for*, analizando las trazas de la siguiente relación de entrenamiento negativa.

2. *Caso 2.* Veamos ahora qué pasaría si no existiese una c con esas características. Siguiendo el algoritmo, entramos en el *if* y elegimos $h \in C(G^*)$ tal que $h \to [b]$ y $h \not\to [a]$.
   Nótese que siempre podemos elegir esa constante, pues de lo contrario, $h \to [b] \implies h \to [a], \forall h \in C(G^*)$, y en particular, $[b] \to [b] \implies [b] \to [a]$, pero esto es una contradicción pues en el preprocesamiento inicial (apartado 3.1) nos hemos asegurado de que $\forall a \not\leq b \in R^-$ se tiene que $[b] \not\to [a]$.
   En el grafo G^* añadimos un nuevo átomo $\zeta \in A(G^*)$ y arco $\zeta \to h$. Tomando $GL^a(b) = \{\phi_1, \ldots, \phi_n\}$, con $\phi_i \to b, \forall i \in \{1, \ldots, n\}$, entonces $\forall i \in \{1, \ldots, n\}$, $[b] \to [\phi_i]$ y como $\zeta \to h \to [b] \to [\phi_i]$, entonces $\zeta \to [\phi_i], \forall i \in \{1, \ldots, n\}$, y por tanto, $\zeta \in Tr(b)$.
   Nótese que al añadir ζ en G^* sólo hemos incluido el arco $\zeta \to h$, eso significa que si $\zeta \to [c] \in V^*$ para algún $c \in C(G)$, esto se debe a la propiedad transitividad y, por tanto, existe el arco $h \to [c]$.

Es decir, ahora $a \in C(G)$ es una constante en las condiciones del Caso 1, esto es, $a \to a$ tal que $\zeta \in Tr(b)$ y $\zeta \not\to [a]$ (de lo contrario, por la observación anterior, $h \to [a]$ #), y por tanto en el siguiente paso del *while* y aplicar de nuevo la función *findStronglyDiscriminantConstant* se dará el Caso 1, logrando así forzar la restricción de traza $Tr(b) \not\subseteq Tr(a)$.

□

Obsérvese que lo que hace el algoritmo es reducir la traza de v hasta que las trazas de los elementos de D^- dejan de estar contenidas en ella. De no conseguirlo, añade átomos nuevos a dichos elementos para que así posean un elemento que no esté en $Tr(v)$. Veámoslo con un ejemplo.

Ejemplo. Retomemos nuestro ejemplo de identificación de barra negra vertical, cuyos grafos G y G^* habíamos dejado como en las Figuras 2.6 y 3.1 respectivamente, y apliquémosle Algorithm 1 para conseguir $R^- = \{v \not\leq T_1^-, v \not\leq T_2^-, v \not\leq T_3^-\}$, forzando el cumplimiento de las restricciones de traza negativa, esto es, $Tr(T_i^-) \not\subseteq Tr(v)$, $\forall i$. Empecemos por $v \not\leq T_1^-$. Como:

$$Tr(v) = \bigcap_{\phi \to v} GL^a([\phi]) = GL^a([0]) = \{0^*, \zeta_1, \zeta_2, \zeta_3\},$$

$$Tr(T_1^-) = \bigcap_{\phi \to v} GL^a([\phi]) = GL^a([0]) = \{0^*, \zeta_1, \zeta_2, \zeta_3\}.$$

tenemos en este caso que $Tr(T_1^-) \subseteq Tr(v)$. Luego, entramos en el *if* y llamamos a la función *findStronglyDiscriminantConstant*(v, T_1^-), que toma un átomo cualquiera ζ de $Tr(T_1^-)$ de G^* y elige una constante $c \in V$ tal que $c \to v \wedge \zeta \not\to [c]$.

En nuestro caso se puede elegir, por ejemplo, $\zeta_1 \in Tr(T_1^-)$ y $c = v$, que cumplen: $v \to v \wedge \zeta_1 \not\to [v]$. Añadimos entonces un nuevo átomo ϕ a G y el arco $\phi \to v$, tranformando los grafos de la siguiente forma:

Comprobemos que efectivamente hemos forzado a que se cumple la restricción negativa de traza $Tr(T_1^-) \not\subseteq Tr(v)$.

T_1^- no tiene átomos nuevos en G, por tanto su traza se mantiene invariante:

$$Tr(T_1^-) = \{0^*, \zeta_1, \zeta_2, \zeta_3\},$$

$$Tr(v) = \bigcap_{\psi \to v} GL^a([\psi]) = GL^a([0]) \cap GL^a([\phi]) = GL^a([\phi]) = \{0^*\}.$$

Por tanto $Tr(T_1^-) \not\subseteq Tr(v)$. Volviendo al flujo del algoritmo, salimos del *while* y del *if* y empezamos el segundo paso del *for* con la relación $v \leq T_2^-$.

Como $Tr(T_2^-) = \{0^*, \zeta_1, \zeta_2, \zeta_3\} \not\subseteq \{0^*\} = Tr(v)$, es decir, se da la restricción negativa de traza, no se entra dentro del *if*.

Pasamos al tercer y último paso del *for*, en el que también se verifica la restricción negativa de traza $Tr(T_3^-) \not\subseteq Tr(v)$.

Por tanto, acabaríamos el algoritmo, con los grafos como en la Figura 3.2, donde podemos observar que se cumplen las relaciones negativas $v \leq T_i^-$ $(1 \leq i \leq 3)$ en G con el orden (2.4).

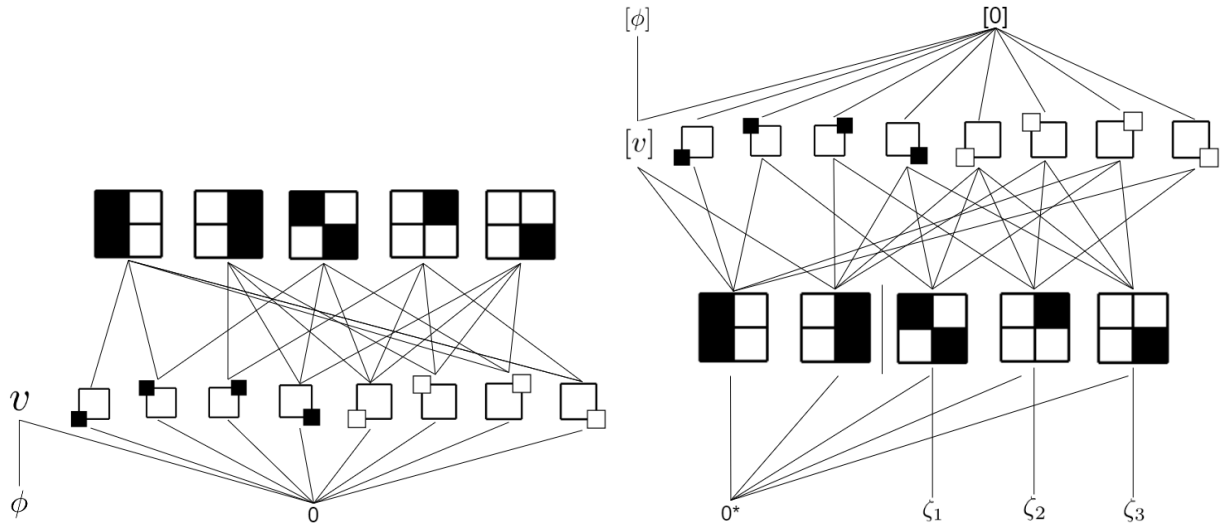

Figura 3.2
Grafos G y G^* en Algorithm 1.

3.4. Obtención de restricciones positivas de traza

Del lema 3.2 dedujimos que $Tr(T_i^+) \subseteq Tr(v)$ es una condición necesaria para que $v \leq T_i^+$ para todo $T_i^+ \in D^+$, por tanto aplicaremos otro algoritmo que conseguirá esta condición. El input es $(d \leq e) \in R^+$, con $d \in C(G)$ y $e \in D^+$:

for $(d \leq e) \in R^+$ **do**
  while $Tr(e) \not\subseteq Tr(d)$ **do**
    Elegimos un átomo $\zeta \in Tr(e) \backslash Tr(d)$;
    Calculamos $\Gamma(\zeta, e) \equiv \{c \in GL^c(e)/\zeta \notin GL([c])\}$;
    if $\Gamma(\zeta, e) = \emptyset$ **then**
      Añadimos el arco $\zeta \to [d]$ en G^*
    end
    else
      Elegimos $c \in \Gamma(\zeta, e)$ aleatoriamente;
      Añadimos un nuevo átomo ϕ a G y el arco $\phi \to c$;
    end
  end
end

Algorithm 2: Restricciones positivas de traza.

Proposición 3.5. *Sean $d \in C(G)$ y $e \in D^+$ tales que $Tr(e) \not\subseteq Tr(d)$. Tras aplicar Algorithm 2, se cumple que $Tr(e) \subseteq Tr(d)$.*

Demostración. Sean $d \in C(G)$ y $e \in D^+$ tales que $Tr(e) \not\subseteq Tr(d)$. Tomamos $\zeta \in Tr(e) \backslash Tr(d)$ y definimos $\Gamma(\zeta, e) = \{c \in GL^c(e)/\zeta \not\to [c]\}$.

Distinguimos 2 casos posibles:

1. Si $\Gamma(\zeta, e) = \emptyset$, añadimos el arco $\zeta \to [d]$ en G^*. Por tanto, $\zeta \in GL^a([d]) \subseteq Tr(d)$.
2. si $\Gamma(\zeta, e) \neq \emptyset$, tomamos $c \in \Gamma(\zeta, e)$. Añadiendo un nuevo átomo $\phi \in A(G)$: $\phi \to c$, conseguimos cambiar la traza de e, ya que

$$\phi \to c \to e \implies Tr(e) \subseteq Tr(\phi) = GL([\phi]),$$

y al estar ϕ únicamente anexado a c se tiene que $GL([\phi]) = GL([c])$. Por tanto,

$$\forall \epsilon \in Tr(e) \implies \epsilon \in GL^a([c]),$$

y de ahí, como $\zeta \not\to [c]$, entonces $\implies \zeta \notin Tr(e)$.

Al salir del *if* volvemos al *while* para volver a comprobar $Tr(e) \not\subseteq Tr(d)$, donde ya no encontrará el átomo ζ del primer paso, pues en el Caso 1, ya habíamos conseguido que $\zeta \in Tr(d)$ y en el Caso 2, ni siquiera ζ está en $Tr(e)$. Asimismo, en ninguno de los dos casos, el cardinal de $Tr(e)$ aumenta por lo que aseguramos que el algoritmo (el bucle *while*) termina en un número finito de veces. □

Obsérvese que durante Algorithm 2 se modifican $GL^a(c)$ para algunas constantes c y por tanto podría ocurrir que alguna restricción negativa $Tr(b) \not\subseteq Tr(a)$ forzada en el Algorithm 1 haya dejado de cumplirse, lo que nos llevaría a tener que aplicar el Algorithm 1 otra vez, y a su vez podría entonces no cumplirse una restricción obtenida con el Algorithm 2. En [10] los autores demuestran el siguiente resultado:

Proposición 3.6. *Es posible forzar las restricciones de traza, tanto negativas como positivas, aplicando Algorithm 1 y 2 un número finito de veces.*

Ejemplo. Sigamos con el ejemplo de la identificación de la barra vertical, aplicándole Algorithm 2, para conseguir $Tr(T_1^+) \subseteq Tr(v) \wedge Tr(T_2^+) \subseteq Tr(v)$.

Calculemos sus trazas a partir de los grafos obtenidos en el paso anterior (véase Figura 3.2):

$$Tr(T_1^+) = Tr(T_2^+) = GL^a([0]) = \{0^*, \zeta_1, \zeta_2, \zeta_3\},$$

$$Tr(v) = GL^a([0]) \cap GL^a([\phi]) = \{0^*\}.$$

Entramos a Algorithm 2 con la primera relación: $v \leq T_1^+$, y debemos elegir un átomo de $Tr(T_1^+) \backslash Tr(v)$. Tomando ζ_1, construimos el conjunto

$$\Gamma(\zeta_1, T_1^+) = \{c \in GL^c(T_1^+)/\zeta_1 \notin GL([c])\} = \{c_1, c_8\}.$$

Elegimos una constante aleatoria de este conjunto, por ejemplo c_1, y añadimos en G nuevo átomo $\epsilon_1 \in V$ y el arco $\epsilon_1 \to c_1$, que también reflejamos en G^*. Al

recalcular las trazas, y teniendo en cuenta que no hemos añadido arco desde ϵ_1 a v, se manteniene $Tr(v)$, esto es:

$$Tr(v) = \{0^*\},\ Tr(T_1^+) = GL^a([\epsilon_1]) = \{0^*\} \implies Tr(T_1^+) \subseteq Tr(v).$$

Salimos del *while* y tomamos la relación $v \leq T_2^+$, en la que

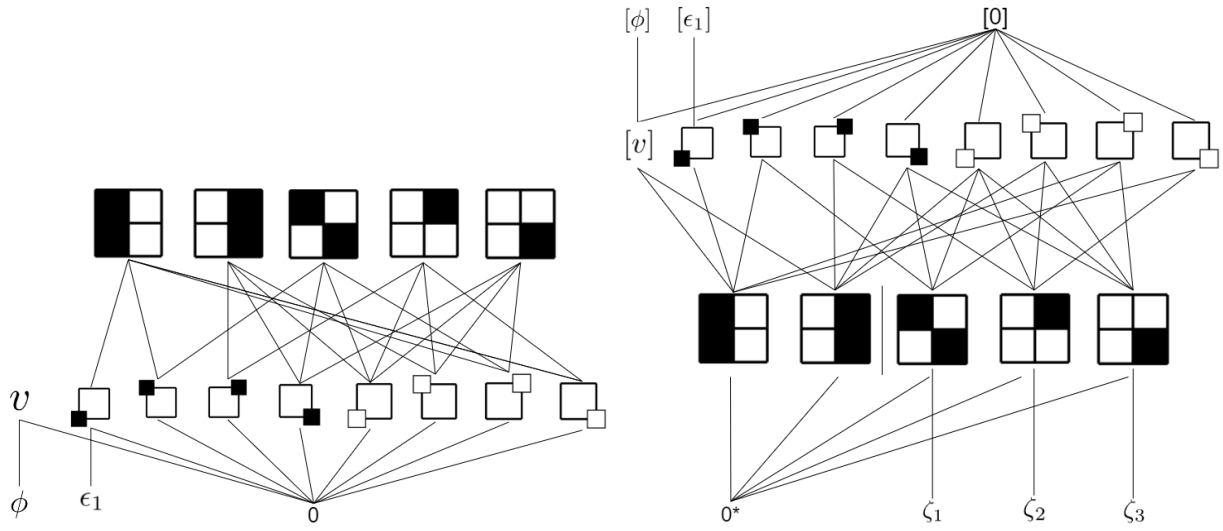

Figura 3.3
Grafos G y G^* tras procesar la relación $v \leq T_1^+$ en Algorithm 2.

$$Tr(T_2^+) = \{0^*, \zeta_1, \zeta_2, \zeta_3\} \not\subseteq \{0^*\} = Tr(v).$$

De nuevo, elegimos un átomo de $Tr(T_2^+) \backslash Tr(v)$, por ejemplo, podemos elegir nuevamente ζ_1, con el que construimos

$$\Gamma(\zeta_1, T_2^+) = \{c \in GL^c(T_2^+)/\zeta_1 \notin GL([c])\} = \{c_3, c_6\}.$$

Elegimos c_3 y añadimos un nuevo átomo ϵ_2 en G y $\epsilon_2 \to c_3$. Recalculamos trazas:

$$Tr(T_2^+) = GL^a([\epsilon_2]) = \{0^*, \zeta_2\}.$$

Sigue sin cumplirse que $Tr(T_2^+) \subseteq Tr(v)$, así que entramos nuevamente en el *while*. Eligiendo $\zeta_2 \in Tr(T_2^+) \backslash Tr(v)$, $\Gamma(\zeta_2, T_2^+) = \{c_4\}$. Añadimos un nuevo átomo ϵ_3 en G con el arco $\epsilon_3 \to c_4$, y finalmente obtenemos

$$Tr(T_2^+) = GL^a([0]) \cap GL^a([\epsilon_2]) \cap GL^a([\epsilon_3]) = \{0^*, \zeta_1, \zeta_2, \zeta_3\} \cap \{0^*, \zeta_2\} \cap \{0^*, \zeta_1, \zeta_3\}.$$

Luego, $Tr(T_2^+) = \{0^*\} \subseteq Tr(v)$. Con ello, termina el algoritmo con los grafos mostrados en la Figura 3.4, cumpliéndose todas las restricciones positivas de traza.

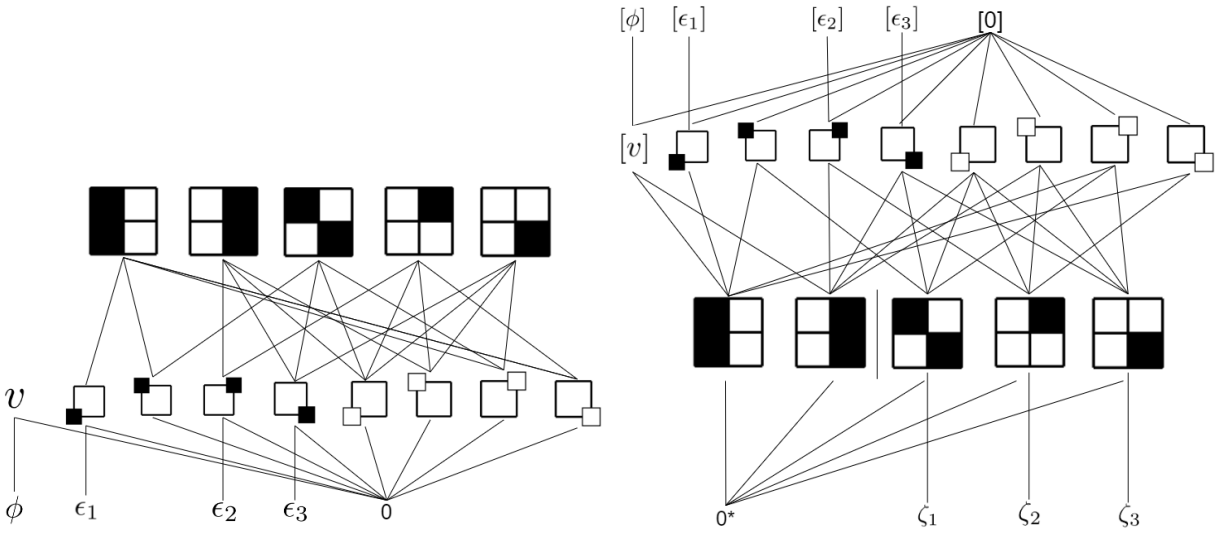

Figura 3.4
Grafos G y G^* tras pasar por Algorithm 2.

3.5. Sparse crossing: obtención de relaciones positivas

Una vez hemos terminado de forzar las relaciones de traza, las relaciones R^- ya se cumplen, vamos a aplicar ahora un tercer algoritmo, denominado *Sparse crossing* (Algorithm 3), aplicable todo $(a \leq b) \in R^+$, con $a \in C(G)$ y $b \in D^+$, que fuerza que se cumplan las relaciones R^+. Antes de proseguir con el proceso,

for $a \leq b \in R^+$ **do**
  Calcular $A \equiv dis(a,b) \equiv GL^a(a) \backslash GL^a(b)$;
  for $\phi \in A$ **do**
    Inicializamos $U \equiv \emptyset, B \equiv GL^a(b) \backslash \{0\}, \Delta \equiv A(G^*) \backslash GL([\phi])$;
    while $\Delta \neq \emptyset$ **do**
      Elegimos un átomo $\epsilon \in B$ aleatoriamente;
      Calculamos $\Delta' \equiv \Delta \cap GL([\epsilon])$;
      if $(\Delta' \neq \Delta) \vee (\Delta = \emptyset)$ **then**
        Creamos un nuevo átomo ψ en G y arcos $\psi \to \phi$ y $\psi \to \epsilon$;
      end
      Sustituimos Δ por Δ';
      Añadimos ϵ a U
    end
    Eliminamos ϵ de B;
  end
end
for $\epsilon \in U$ **do**
  Creamos un nuevo átomo ϵ' en G y el arco $\epsilon' \to \epsilon$
end
Eliminamos de G todos los elementos de $U \cup A$ de G y sus correspondientes en G^*

Algorithm 3: Sparse crossing.

vamos a demostrar un par de lemas que nos servirán para probar que Algorithmo 3 cumple con su cometido.

Observación: En el ciclo *for* final de este algoritmo proponemos como opción de optimización comprobar si la traza se mantiene, para así evitar añadir átomos nuevos innecesariamente. Así será como se aplicará en el ejemplo.

Lema 3.7. *Sean* $a, b \in V : a \not\leq b$ *y sea* $\phi \in dis(a,b) = GL^a(a) \backslash GL^a(b)$. *Sea* $\Delta = A(G^*) \backslash Tr(\phi)$ *y para* $\epsilon \in GL^a(b)$, *sea* $\Delta' = \Delta \cap GL^a([\epsilon])$. *Entonces:*

$$Tr(b) \subseteq Tr(a) \implies \exists \epsilon \in GL^a(b) : \ (\Delta \neq \Delta') \vee (\Delta = \Delta' = \emptyset).$$

Demostración. Si $\Delta = \emptyset$ es claro que $\Delta = \Delta' = \emptyset$. Supongamos ahora que $\Delta \neq \emptyset$. Por reducción al absurdo, establezcamos que

$$\forall \epsilon \in GL^a(b), \ \Delta = \Delta' = \Delta \cap GL^a([\epsilon]) \neq \emptyset,$$

Por tanto, $\forall \epsilon \in GL^a(b)$, $\forall \zeta \in \Delta$, se tiene que $\zeta \in GL^a([\epsilon])$. De aquí,

$$\forall \zeta \in A(G^*) \backslash Tr(\phi), \ \zeta \in Tr(b),$$

pero, por hipótesis, $Tr(b) \subseteq Tr(a)$, luego $\forall \zeta \in A(G^*) \backslash Tr(\phi)$:

$$\zeta \in Tr(a) = \bigcap_{\varphi \to a} GL^a([\varphi]) \subseteq GL^a([\phi]) \implies \zeta \in GL^a([\phi]) = Tr(\phi).$$

Estamos en el caso de que $\Delta \neq \emptyset$, luego $\exists \zeta \in \Delta = A(G^*) \backslash Tr(\phi)$, pero tal ζ, por lo anterior, cumple también que $\zeta \in Tr(\phi)$. # □

Lema 3.8. *Sean* $a, b \in V : a \not\leq b$ *y sea* $\phi \in dis(a,b) = GL^a([a]) \backslash GL^a([b])$. *Definimos* $\Delta = A(G^*) \backslash Tr(\phi)$. *Tomando* $\{\epsilon_1, \ldots, \epsilon_{n-1}\} \subseteq GL^a(b)$, *se tiene que, habiendo añadido nuevos átomos* $\{\varphi_1, \ldots, \varphi_{n-1}\} \subseteq A(G)$ *con arcos* $\varphi_i \to \phi \wedge \varphi_i \to \epsilon_i$ *si* $Tr(b) \subseteq Tr(a) \wedge \Delta \cap (\bigcap_{i=1}^{n-1} GL^a([\epsilon_i])) \neq \emptyset$, *entonces:*

$$\exists \epsilon_n \in GL^a(b) : \Delta \cap (\bigcap_{i=1}^{n-1} GL^a([\epsilon_i])) \supsetneq \Delta \cap (\bigcap_{i=1}^{n} GL^a([\epsilon_i])).$$

Demostración. Tras haber añadido los nuevos átomos $\varphi_i \in A(G)$, $1 \leq i \leq n$,

$$Tr(\phi) = \bigcap_{i=1}^{n-1} GL^a([\varphi_i]) = GL^a([\phi]) \cup (\bigcap_{i=1}^{n-1} GL^a([\epsilon_i])),$$

Veamos que $\exists \epsilon_n \in GL^a(b) : \Delta \cap (\bigcap_{i=1}^{n-1} GL^a([\epsilon_i])) \neq \Delta \cap (\bigcap_{i=1}^{n} GL^a([\epsilon_i]))$.

Por reducción al absurdo, supongamos que se cumple la igualdad, esto es,

$$\forall \epsilon_n \in GL^a(b) \backslash \{\epsilon_1, \ldots, \epsilon_{n-1}\}, \ \ \forall \zeta \in \Delta \cap (\bigcap_{i=1}^{n-1} GL^a([\epsilon_i])) \implies \zeta \to [\epsilon_n].$$

Luego,

$$\forall \epsilon_n \in GL^a(b) \backslash \{\epsilon_1, \ldots, \epsilon_{n-1}\}, \quad \forall \zeta \in \Delta \cap (\bigcap_{i=1}^{n-1} GL^a([\epsilon_i])) \implies \zeta \in GL^a([\epsilon_n]),$$

y como $Tr(b) \subseteq Tr(a)$, entonces:

$$\forall \zeta \in \Delta \cap (\bigcap_{i=1}^{n-1} GL^a([\epsilon_i])), \quad \zeta \in Tr(b) \subseteq Tr(a) = \bigcap_{\psi \to a} GL^a([\varphi]) \subseteq GL^a([\phi]), \text{ pero}$$

$$\zeta \in \Delta \cap (\bigcap_{i=1}^{n-1} GL^a([\epsilon_i])) = \bigcap_{i=1}^{n-1} GL^a([\epsilon_i]) \backslash GL^a([\phi]) \implies \zeta \notin GL^a([\phi]) \ \#.$$

□

Proposición 3.9. *Sean $a, b \in V : a \not\leq b \ \wedge Tr(b) \subset Tr(a)$. Después de aplicar Algorithm 3, se tiene que $a \leq b$ en G (véase orden en G en fórmula (2.4)).*

Demostración. Sea $\phi \in dis(a, b)$, con $\Delta_0 = A(G^*) \backslash GL^a([\phi])$ entramos en el *while*. Veamos que el bucle siempre termina siempre.

Por el Lema 3.7, $\exists \epsilon_1 \in GL^a(b) : \Delta_1 = \Delta_0 \cap GL^a([\epsilon_1]) \neq \Delta_0$ y por tanto en el primer ciclo del *if* siempre va a activarse. Cuando esto ocurre, Δ (en este caso lo hemos denotado Δ_0) es sustituido por el conjunto que el algoritmo llama Δ' (en este caso, Δ_1).

Para hacer la prueba más clara, vamos a denotar por Δ_i al conjunto Δ en el ciclo i-ésimo, considerando el ciclo inicial como ciclo 0. El lema 3.8 nos garantiza que, en el paso n-ésimo, siempre va a existir un $\epsilon_n \in GL^a(b) : \Delta_{n-1} \neq \Delta_{n-1} \cap GL^a([\epsilon_n]) := \Delta_n$ y por tanto se activará el condicional *if* del algoritmo. En general, para $n \in \mathbb{N}$, si $\Delta_n \neq \emptyset \implies \Delta_{n+1} \subsetneq \Delta_n$, y como omo Δ_0 es finito, podemos asegurar $\exists n_0 : \Delta_{n_0} = \emptyset$, terminando así el *while* para el átomo ϕ.

Veamos ahora que al terminar Algorithm 3, se consigue que $a \leq b$. En el ciclo n-ésimo añadimos un nuevo átomo $\psi_n \in V$ y los arcos $\psi_n \to \phi \wedge \psi_n \to \epsilon_n$. Al acabar el bucle *for* se tiene que $\phi = \bigodot_{i=1}^{n} \psi_i$, donde $\psi_i \to b$, Al eliminar ϕ del grafo G, lo hemos sustituido por n átomos que, a diferencia de ϕ, sí están dirigidos a b, y por tanto, hemos reducido el cardinal de $dis(a, b)$ en 1 elemento.

Al replicar este algoritmo con los demás átomos de $dis(a, b)$ obtenemos finalmente que $GL^a(a) \subseteq GL^a(b)$, luego $a \leq b$ en G. □

Proposición 3.10. *Algorithm 3 deja la traza de los átomos invariante.*

Demostración. Al terminar el proceso del Algorithm 3, se tiene que $\phi = \bigodot_{i=1}^{n} \varphi_i$ y que $\Delta_n = \Delta \cap (\bigcap_{i=1}^{n} GL^a([\epsilon_i])) = \emptyset$. Veamos que

$$Tr_0(\phi) = Tr_1(\phi) = Tr(\bigodot_{i=1}^{n} \varphi_i),$$

donde Tr_0 denota la traza antes del algoritmo y Tr_1, después.

" $\supseteq$ " Sea $\zeta \in Tr_1(\phi) = Tr(\bigodot_{i=1}^{n} \varphi_i)) = \bigcap_{i=1}^{n} Tr(\varphi_i)$. En particular, $\zeta \in Tr(\varphi_i), \forall i \in \{1, \ldots, n\}$, y como $Tr(\varphi_i) = GL^a([\phi]) \cup GL^a([\epsilon_i])$, tenemos que

$$\zeta \in \bigcap_{i=1}^{n} Tr(\varphi_i) = \bigcap_{i=1}^{n} (GL^a([\phi]) \cup GL^a([\epsilon_i])) = GL^a([\phi]) \cup (\bigcap_{i=1}^{n} GL^a([\epsilon_i])).$$

Por reducción al absurdo, supongamos que $\exists \zeta \in \bigcap_{i=1}^{n} Tr(\varphi_i) : \zeta \notin Tr_0(\phi) = GL^a([\phi])$, entonces:

$$\zeta \in (GL^a([\phi]) \cup (\bigcap_{i=1}^{n} GL^a([\epsilon_i])) \backslash GL^a([\phi]) = \bigcap_{i=1}^{n} GL^a([\epsilon_i]) \backslash Tr(\phi) =$$

$$(A(G^*) \backslash Tr(\phi)) \cap \bigcap_{i=1}^{n} GL^a([\epsilon_i]) = \Delta \cap \bigcap_{i=1}^{n} GL^a([\epsilon_i]) = \Delta_n$$

pero $\Delta_n = \emptyset$ por hipótesis, por lo que estamos ante una contradicción.

" $\subseteq$ " Veamos ahora que $Tr_0(\phi) \subseteq \bigcap_{i=1}^{n} Tr(\varphi_i)$.
Para todo $i \in \{1, \ldots, n\}$, tenemos $\varphi_i \to \phi$, luego,

$$Tr(\phi) = GL^a([\phi]) \subseteq GL^a([\varphi_i]) = Tr(\varphi_i),$$

y así podemos concluir que $Tr(\phi) \subseteq \bigcap_{i=1}^{n} Tr(\varphi_i)$. □

Hemos probado que con Algorithm 3 logramos obtener las relaciones R^+, y además, que deja las trazas de cualquier elemento invariante, manteniéndose por tanto, las relaciones R^- que conseguimos en los pasos anteriores.

En este punto del método de AML ya se cumplen todas las relaciones de entrenamiento que buscábamos, sin embargo, vamos a utilizar un par de algoritmos más que explicaremos en los próximos apartados.

Ejemplo. Retomemos el ejemplo trabajado en Figura 3.4. Queremos conseguir que $v \leq T_1^+ \wedge v \leq T_2^+$. En el momento actual se tiene que

$$GL^a(v) = \{0, \phi\}, \quad GL^a(T_1^+) = \{0, \epsilon_1\}, \quad GL^a(T_2^+) = \{0, \epsilon_2, \epsilon_3\}.$$

No se da ninguna de las desigualdades buscadas, realizamos por tanto primero el *"crossing"* entre v y T_1^+. Elegimos un átomo en $dis(v, T_1^+)$, siendo en este caso nuestra única opción el átomo ϕ.

Sea $\Delta = A(G^*) \backslash GL([\phi])$, y busquemos un átomo ϵ en $GL^a(T_1^+)$. Entonces, $\Delta' = \Delta \cap GL([\epsilon]) \neq \Delta$. Vamos a tomar $\epsilon_1 \in V$, quedándonos:

$$\Delta = A(G^*) \backslash GL([\phi]) = \{\zeta_1, \zeta_2, \zeta_3\}, \quad \Delta' = \Delta \cap GL([\epsilon_1]) = \emptyset.$$

Se tiene que $\Delta \neq \Delta'$ así que creamos un nuevo átomo ϕ_1 en G con arcos $\phi_1 \to \phi$, $\phi_1 \to \epsilon_1$. Como $\Delta = \emptyset$, termina el bucle. La traza de ϵ_1 no ha cambiado así que no es necesario añadir un nuevo átomo para fijarla. Eliminamos los átomos ϕ y ϵ y obtenemos los grafos de las Figuras 3.5 y 3.6. En este punto

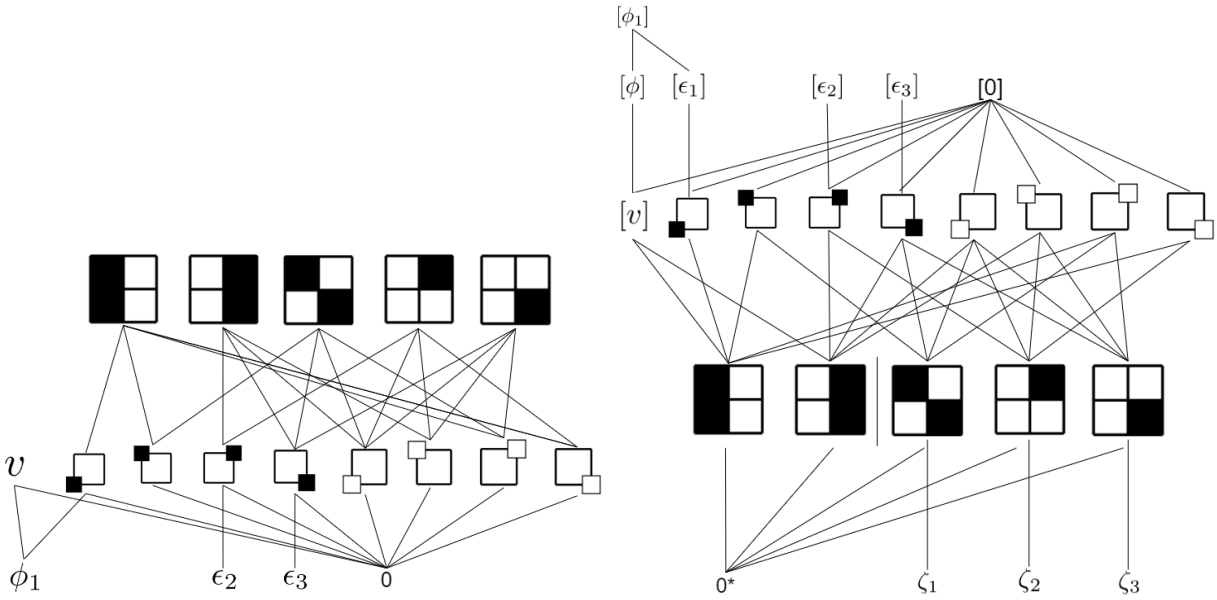

Figura 3.5
Grafos G y G^* tras incorporar ϕ_1 en Algorithm 3.

tenemos que

$$GL^a(v) = \{0, \phi_1\}, \quad GL^a(T_1^+) = \{0, \epsilon_1', \phi_1\},$$

es decir, se tiene que $v \leq T_1^+$, así que sólo nos falta conseguir $v \leq T_2^+$.

Entramos de nuevo en el bucle *for*, eligiendo un átomo de $dis(v, T_2^+)$. El único que tenemos es ϕ_1, y tomamos un átomo de $B = GL^a(T_2^+) = \{\epsilon_2, \epsilon_3\}$, por ejemplo ϵ_2. En este caso, nos queda:

$$\Delta = A(G^*) \backslash GL([\phi_1]) = \{\zeta_1, \zeta_2, \zeta_3\}, \quad \Delta' = \Delta \cap GL([\epsilon_2]) = \{\zeta_2\}.$$

Como son distintos, añadimos un nuevo átomo ϕ_2 a G, con $\phi_2 \to \phi_1, \phi_2 \to \epsilon_2$. Tenemos ahora que $\Delta = \{\zeta_2\}$. Elegimos otro átomo de $B = GL^a(T_2^+)$, esto es, ϵ_3, siendo $\Delta' = \Delta \cap GL^a([\epsilon_3]) = \emptyset$. Añadimos un nuevo átomo ϕ_3 a G con arcos $\phi_3 \to \phi_1, \phi_3 \to \epsilon_3$. Como las trazas de ϵ_2 y ϵ_3 se han mantenido constantes no es necesario añadir nuevos átomos para fijarlas, y eliminando los átomos $\phi_1, \epsilon_2, \epsilon_3$, obtenemos los grafos de las Figuras 3.7 y 3.8.

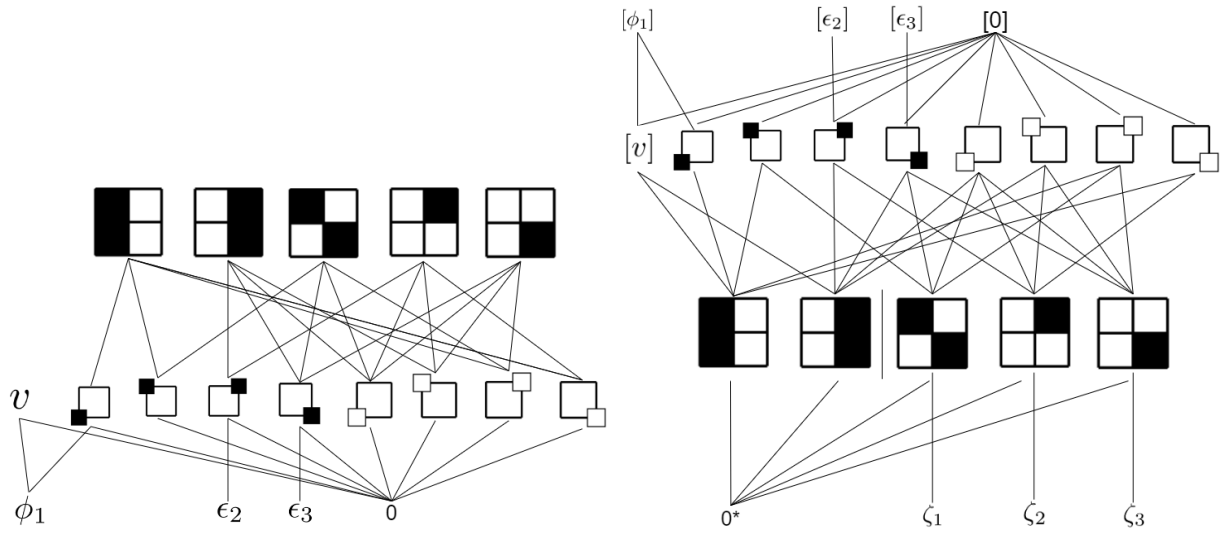

Figura 3.6
Grafos G y G^* tras eliminar los átomos $[\phi]$ y $[\epsilon_1]$ en Algorithm 3.

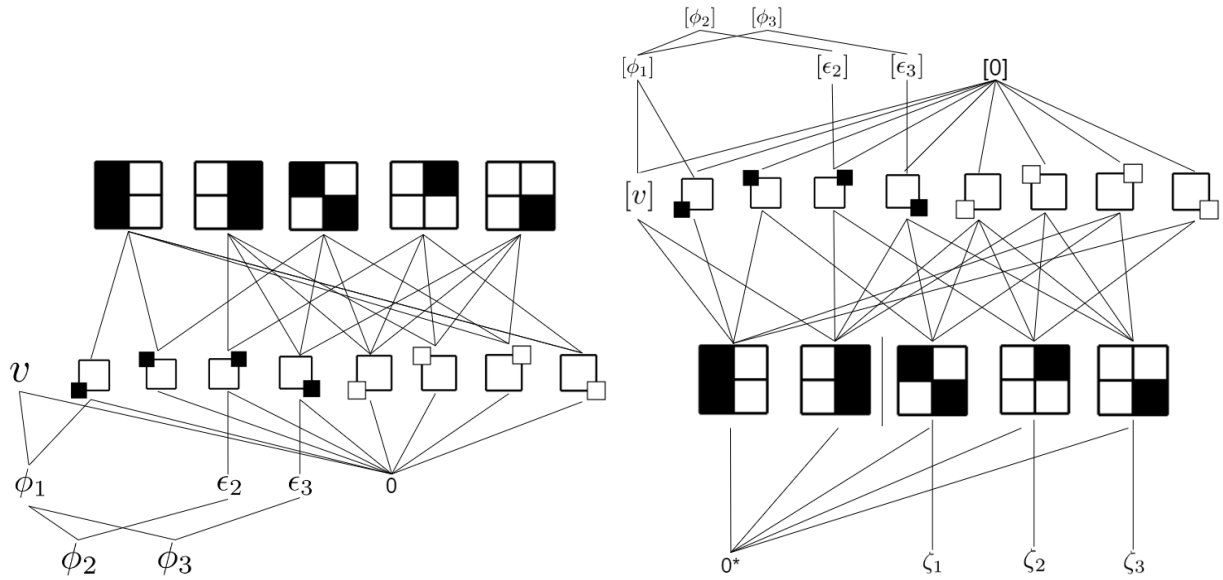

Figura 3.7
Adición de los átomos ϕ_2 y ϕ_3 en G y sus correspondientes en G^* en Algorithm 3.

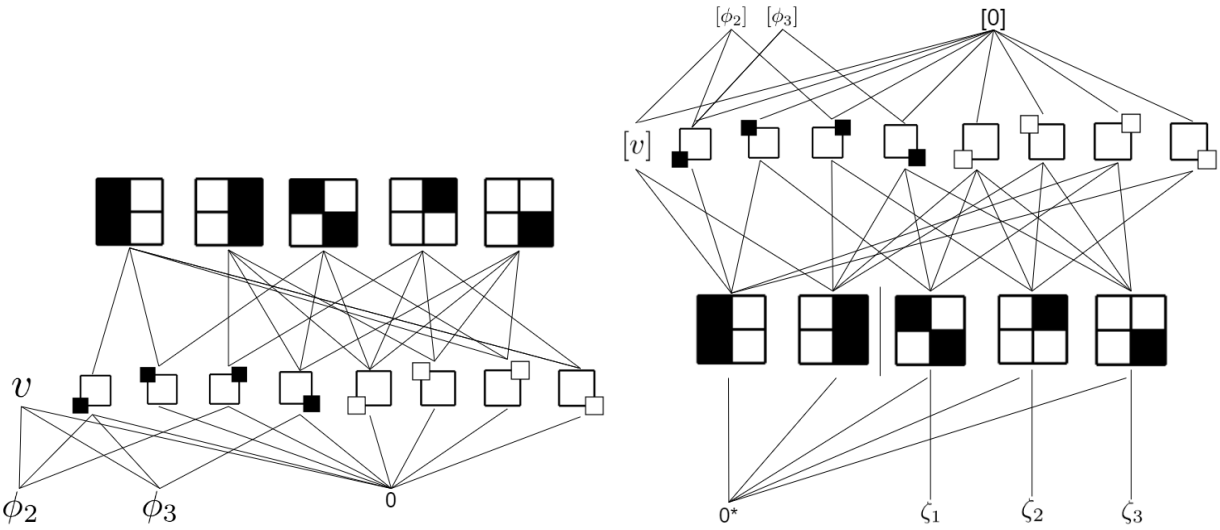

Figura 3.8
Eliminación de los átomos ϕ_1, ϵ_2 y ϵ_3 de G y sus correspondientes en G^* en Algorithm 3.

Con lo anterior, finalmente obtenemos:

$$GL^a(v) = \{0, \phi_2, \phi_3\},\ GL^a(T_1^+) = \{0, \phi_2, \phi_3\},\ GL^a(T_2^+) = \{0, \phi_2, \phi_3\},$$

es decir, en este punto ya se cumplen todas las relaciones de entrenamiento positivas: $v \leq T_i^+, i = 1, 2$.

Por tanto nuestro algoritmo ya "ha aprendido" de los datos de entrenamiento D, y es capaz de hacer predicciones sobre el cumplimiento de tal propiedad para todo $T \in \omega$, esto es, sí $v \leq T$, tal y como lo formulamos en el apartado 2.9.

3.6. Reducción del número de átomos de G y G^*

En esta etapa del proceso ya tenemos un modelo que satisface todas las relaciones de entrenamiento R. Sin embargo, se probará en secciones posteriores que la precisión del modelo obtenido está inversamente relacionada con el número de átomos, por tanto vamos a aplicar otro algoritmo que reduce el cardinal de $A(G)$, manteniendo invariantes las relaciones de traza. Este algoritmo (Algorithm 4), sin embargo, sí que puede alterar las relaciones positivas, por tanto es ideal aplicarlo justo antes de usar el *Sparse crossing* (Algorithm 3).

Lema 3.11. *Sean* $b, c \in V$, $Tr(b \odot c) = Tr(b) \cap Tr(c)$.

Demostración. Usando las definiciones detalladas en el apartado 2.5,

$$Tr(b \odot c) = Tr(GL^a(b) \cup GL^a(c)) = \bigcap_{\phi \to b \vee \phi \to c} GL^a([\phi]) =$$

$$(\bigcap_{\phi \to b} GL^a([\phi])) \cap (\bigcap_{\psi \to c} GL^a([\psi])) = Tr(b) \cap Tr(c).$$

□

En consecuencia, para mantener invariante la traza de los términos, es suficiente con mantener la traza de las constantes. Algoritm 4 calcula un subconjunto de átomos suficiente para mantener las trazas de las constantes invariantes.

Inicializamos los conjuntos $Q \equiv \emptyset$, $\Delta = C(G)$;
while $\Delta \neq \emptyset$ **do**
  Elegimos $c \in \Delta$ aleatoriamente y lo eliminamos de Δ;
  Calculamos $S_c \equiv Q \cap GL(c)$;
  if $S_c = \emptyset$ **then**
    definimos $W_c \equiv A(G^*)$;
  end
  else
    Calculamos $W_c \equiv \bigcap_{\phi \in S_c} GL^a([\phi])$;
  end
  Calculamos $\Phi_c \equiv \{[\phi]/\phi \in GL^a(c)\}$;
  while $W_c \neq Tr(c)$ **do**
    Elegimos un átomo $\xi \in W_c \backslash Tr(c)$ aleatoriamente;
    Elegimos un átomo ϕ tal que $[\phi] \in \Phi_c \backslash GU(\xi)$ aleatoriamente;
    Añadimos ϕ a Q ;
    Sustituimos W_c por $W_c \cap GL^a([\phi])$;
  end
end
Eliminamos todos los átomos del conjunto $A(G) \backslash Q$.

Algorithm 4: Reducción del número de átomos

Proposición 3.12. *Algorithm 4 no altera las trazas de las constantes.*

Demostración. Como vemos en el algoritmo, el papel que juega Q es ir almacenando en cada paso los átomos que son imprescindibles para mantener todas las trazas de las constantes, inicializándose como conjunto vacío. El procedimiento va recorriendo todas las constantes $c \in C(G)$, almacenando como W_c la traza de c *actualizada en cada paso*, es decir, $W_c = \bigcap_{\phi \in Q \cap GL(c)} GL^a([\phi])$.

De ahí, está claro que el proceso siempre termina en un número finito de pasos. En el peor de los casos, para todas las constantes c, el bucle *while* interno añadirá todos los átomos que tenía inicialmente $GL^a(c)$ en G.

Por otro lado, es trivial que las trazas de todas las constantes quedan invariantes, pues el *while* interno finaliza cuando se alcanza que $W_c = Tr(c)$. □

Ejemplo. En nuestro ejemplo, antes de aplicar el *Sparse Crossing* (véase Figura 3.8), solo teníamos 5 átomos siendo todos necesarios para mantener las trazas. En este caso no haría falta aplicar Algorithm 4, sin embargo, para ilustrar su

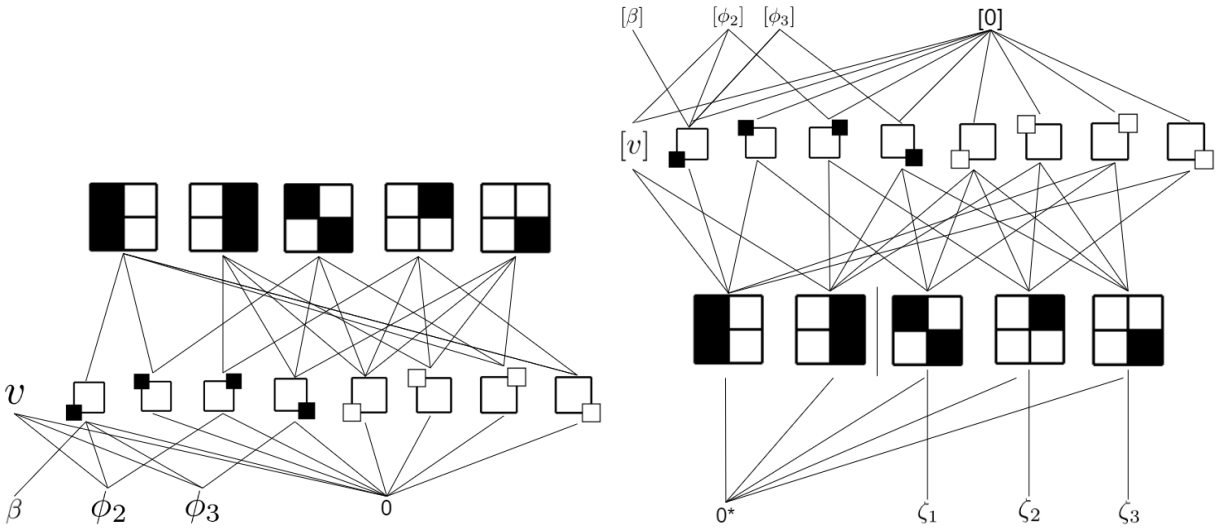

Figura 3.9
Caso de estudio de Algorithm 4 para los grafos G y G^*.

funcionamiento, vamos estudiar el referido grafo, donde hemos añadido un átomo $\beta \in V$ y un arco $\beta \to c_1$, y sus correspondientes en G^* (Figura 3.9).

Este algoritmo va eligiendo subconjuntos de átomos necesarios para mantener las trazas de las constantes invariantes, y para ello, para cada constante, va cogiendo aletoriamente los átomos que cumplan tal condición. Así, por ejemplo, para la constante c_1, podemos mantener su traza manteniendo sólo β o manteniendo ϕ_2 y ϕ_3.

Sin embargo, para mantener la traza de c_3, necesitamos tener ϕ_2 y para la traza de c_4 necesitamos ϕ_3, por tanto si el algoritmo al comprobar c_1 decide mantener β, al final como buscamos mantener todas las trazas no conseguiremos eliminar ningún átomo. Por el contrario, si el algoritmo elige mantener ϕ_2 y ϕ_3 para c_1, o comprueba primero las trazas de c_3 y c_4, conseguiremos eliminar el átomo β, que en este caso es redundante.

Como hemos comprobado, este algoritmo no siempre consigue reducir el cardinal de $A(G)$ y cuando lo hace, no siempre es de forma óptima. Por tanto, en los pasos donde apliquemos el algoritmo conviene aplicarlo varias veces y quedarnos con el resultado que tenga menor número de átomos.

3.7. Uso el Modelo AML y evaluación de su calidad

Una vez llegados a este punto, el *Modelo Predictivo de AML* que buscábamos es $\mathcal{M}$, formado por el semirretículo M tras aplicarle los Algoritmos 1, 2, 4

y 3. $\mathcal{M}$ es capaz de predecir si nuevos datos (términos) tienen una barra negra vertical, esto es, si $v \leq T$.

Como se avanzó en el apartado 2.9, para todo $T \in \Omega$, aplicando la codificación realizada en 2.2, sabemos que $\exists c_1, \ldots, c_n \in C(G)$ tales que $T = \bigodot_{i=1}^{n} c_i$. Entonces, T tiene una barra negra vertical sí, y solo si, por la fórmula (2.7),

$$v \leq T \iff GL^a(v) \subseteq \bigcup_{i=1}^{n} GL^a(c_i).$$

Es decir, que la predicción se reduce a la comprobación de una inclusión de conjuntos de átomos del grafo G.

Por otro lado, tras la realización de todo modelo de Machine Learning Supervisado, se debe llevar a cabo un análisis sobre su calidad, aplicando el modelo a un nuevo conjunto de *Datos de Prueba* (etiquetados), D_{test}, y comprobar si los resultados del modelo coinciden con sus respectivas etiquetas.

Para ello se suele usar la *Matriz de Confusión*, formada por:

- *Datos "True Positive":* $TP = \{T \in D_{test} / v \leq T$ y $\mathcal{M}$ predice $v \leq T\}$;
- *Datos "False Positive":* $FP = \{T \in D_{test} / v \not\leq T$ y $\mathcal{M}$ predice $v \leq T\}$;
- *Datos "True Negative":* $TN := \{T \in D_{test} / v \not\leq T$ y $\mathcal{M}$ predice $v \not\leq T\}$;
- *Datos "False negative":* $FN = \{T \in D_{test} / v \leq T$ y $\mathcal{M}$ predice $v \not\leq T\}$.

Algunas de las métricas más usadas para evaluar los modelos son:

- Exactitud (Accuracy): $\frac{|TP|+|TN|}{|TP|+|TN|+|FP|+|FN|}$,
- Precisión (Precision): $\frac{|TP|}{|TP|+|FP|}$,
- Sensibilidad (Sensitivity/Recall): $\frac{|TP|}{|TP|+|FN|}$.

Ejemplo. Veamos un ejemplo de cómo aplicar el Modelo AML $\mathcal{M}$ para predecir un resulado. Sea $T \in \Omega$:

Visualmente está claro que $v \leq T$ puesto que tiene una barra negra vertical. Según el modelo AML, dado que $T = c_1 \odot c_2 \odot c_3 \odot c_8$, de Figura 3.8 se concluye:

$$GL^a(v) = \{0, \phi_2, \phi_3\} \subseteq$$

$$\subseteq GL^a(T) = GL^a(c_1) \cup GL^a(c_2) \cup GL^a(c_3) \cup GL^a(c_8) = \{0, \phi_2, \phi_3\}.$$

Por tanto, $T \in TP$, es decir, el Modelo AML $\mathcal{M}$ precide que T tiene una barra negra vertical.

Siguiendo este procedimiento podemos clasificar un conjunto de datos de prueba D_{test} y calcular la calidad del modelo a través de las métricas elegidas.

Por otro lado, si la calidad es menor que la esperada, es posible entrenar más el modelo, usando el procedimiento descrito en los apartados 3.8 y 3.9.

3.8. Preparación para aprendizaje por lotes: términos y relaciones de fijación

Partiendo de un Modelo $\mathcal{M}$ de AML, y disponiendo de más datos de $D' \in \Omega$, y por tanto, de nuevas relaciones de entrenamiento, R', es posible "aplicar mayor aprendizaje".

En tal caso, para hacer el algoritmo más eficiente desde un punto de vista computacional, se eliminarán previamente de G los términos involucrados en las relaciones de entrenamiento R, pero manteniendo a la vez la "información aprendida" en el primer ciclo de entrenamiento. Para conservar la mayor cantidad posible de dicho aprendizaje se usarán unos nuevos términos llamados *pinning terms*, o *términos de fijación*.

Definición 3.13. *Para cada átomo $\phi \in G$, el término de fijación (o pinnig term) asociado a ϕ se define como el nuevo término: $T_\phi = \bigodot_{\phi \not\to c,\ c \in C(M)} c$ de G, es decir, la unión de todas las constantes no mayores que dicho átomo.*

A partir de los términos de fijación, se definen las relaciones de fijación como el nuevo conjunto: $R_p = \bigcup_{\phi \in A(G)} \{c \not\leq T_\phi \ / \ c \in C(M) : \phi \to c\}$.

Las relaciones de fijación se pueden entender como "hipótesis" que el algoritmo realizará sobre el problema. Durante el aprendizaje estas hipótesis se irán descartando o reforzando usando los nuevos datos.

Algorithm 5 crea los términos y relaciones de fijación partiendo de un Modelo AML ya creado como se ha visto en los apartados anteriores.

for $\phi \in A(G)$ **do**
  Calculamos $H = C(G) \backslash GU(\phi)$;
  Creamos un nuevo término $T_\phi = \bigodot_{c \in H} c$
  Creamos la correspondiente constante $[T_\phi]$ en G^* y la arista $0^* \to [T_\phi]$
  Creamos los arcos $c \to T_\phi, \forall c \in H$ en G
  Y los correspondientes arcos $[T_\phi] \to [c]$ en G^* **for** $c \in C(G) \cap GU(\phi)$ **do**
    Añadimos $(c \not\leq T_\phi)$ a R_p;
  end
end

Algorithm 5: Creación de términos y relaciones de fijación

Ejemplo. Retomando nuestro caso, en la Figura 3.8 quedaron dos átomos (no nulos), ϕ_2 y ϕ_3. Como $GU^c(\phi_2) = \{c_1, c_3\}$, creamos $T_{\phi_2} = c_2 \odot c_4 \odot c_5 \odot c_6 \odot c_7 \odot c_8$, y como $GU^c(\phi_3) = \{c_1, c_4\}$, $T_{\phi_3} = c_2 \odot c_3 \odot c_5 \odot c_6 \odot c_7 \odot c_8$. Al añadir estos dos términos de fijación con sus correspondientes arcos a los grafos G y G^* obtenemos la Figura 3.10.

Las relaciones de fijación que se obtienen son:

$$R_p = \{v \not\leq T_{\phi_2}, c_1 \not\leq T_{\phi_2}, c_3 \not\leq T_{\phi_2}, v \not\leq T_{\phi_3}, c_1 \not\leq T_{\phi_3}, c_4 \not\leq T_{\phi_3}\}.$$

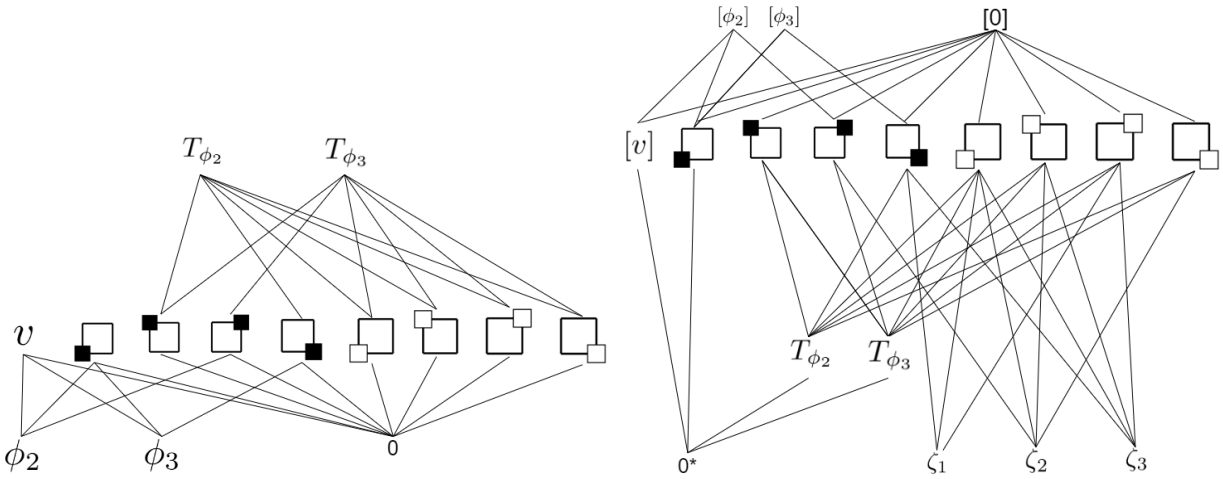

Figura 3.10
Creación de los términos de fijación según Algorithm 5.

Tras la creación de las relaciones de fijación, veremos en el apartado siguiente que antes de iniciar un nuevo ciclo de aprendizaje, se eliminarán de G los términos de D implicados en las relaciones de entrenamiento R, y sus correspondientes imágenes en G^*. Obsérvese en la Figura 3.10 que en G^* se mantienen los arcos desde los átomos $\{\zeta_1, \zeta_2, \zeta_3\}$ hacia las imágenes de las constantes, debido a que G^* es transitivamente cerrado.

3.9. Aprendizaje por lotes: ciclos de entrenamiento

Veremos a continuación cómo realizar un entrenamiento por lotes, donde el algoritmo AML irá aumentando su nivel de aprendizaje a medida que pasa por los sucesivos ciclos de entrenamiento. Lo explicado en el presente capítulo hasta el apartado 3.7 puede verse como el ciclo 0, donde $R_0 = R$. En el ciclo i-ésimo ($i >= 1$), se trabajará con:

$$R_i \cup R_p^{i-1},$$

donde R_i son las relaciones de entrenamiento correspondientes a nuevos datos D_i que se incorporan en el ciclo i, y R_p^{i-1} son las relaciones de fijación, que acumulan la información aprendida hasta el paso $i-1$, y se obtienen al final del apartado $(i-1)$-ésimo, como se explica en el apartado 3.8.

Al iniciarse el ciclo i-ésimo, justo se ha teminado de eliminar en el paso $(i-1)$-ésimo los términos implicados en R_{i-1}, y en primer lugar se añaden como nodos de G y sus correspondientes en G^* los términos de los nuevos datos de entrenamiento D_i y se crean las nuevas relaciones de entrenamiento R_i.

Asimismo, comprobamos que los grafos G y G^* cumplen sus propiedades iniciales (apartados 2.1 y 2.6), incorporando los arcos que fueran necesarios.

Además, se "preprocesan"los nuevos datos como en el apartado 3.1, donde, en vez de considerar sólo R_i, tomamos $R_i \cup R_p^{i-1}$. Así, para cada $(a \leq b) \in R_i \cup R_p^{i-1}$ añadimos el arco $[b] \to [a]$ en G^* (lo que implica que $[b] \leq [a]$) y para cada $(a \not\leq b) \in R_i \cup R_p^{i-1}$ añadimos un nuevo átomo $\xi \to [b]$, de manera que $[b] \not\leq [a]$.

Si se consiguen forzar estas relaciones, los datos son consistentes (de no serlo, alguna relación que debiera ser negativa aparecería como positiva, es decir, para $a \leq b$ se tendría $a \to b$). Si no, haremos:

- Si la relación incosistente está en R_i, significa que los nuevos datos D_i están *corruptos* y por tanto debemos obtener nuevos datos.
- Si la relación inconsistente está en R_p^{i-1}, la eliminamos de R_p^{i-1} y su correspondiente término de G y G^*. Esta acción representa "descartar la hipótesis"que planteaba dicha relación.

Tras comprobar la consistencia, reduciremos el cardinal de $A(G^*)$.

Inicializamos los conjuntos $Q \equiv 0, S \equiv R^-$
while $S \neq \emptyset$ **do**
  Elegimos $r \in S$ aleatoriamente y lo eliminamos de S. Sea $r \equiv (a \not\leq b)$;
  if $(GL^a([b]) \backslash GL^a([a])) \cap Q = \emptyset$ **then**
    Elegimos un átomo $\xi \in GL^a([b]) \backslash GL^a([a])$ y lo añadimos a Q;
  end
end
Eliminamos todos los átomos del conjunto $A(G^*) \backslash Q$;

Algorithm 6: Reducción del cardinal de $A(G^*)$

Lema 3.14. *Sea $a \not\leq b \in R_i^-$, tras aplicar Algorithm 6, se cumple $[b] \not\to [a]$.*

Demostración. El conjunto Q contiene los átomos que vamos a mantener. Empieza siendo vacío e iremos añadiendo los átomos imprescindibles. Sea $(a \not\leq b) \in R_i^-$, por construcción, $[b] \not\leq [a]$, y por Lema 2.9, $GL^a([b]) \not\subseteq GL^a([a])$.

Si $(GL^a([b]) \backslash GL([a])) \cap Q = \emptyset$, tomamos un átomo $\xi \in GL^a([b])$ tal que $\xi \notin GL([a])$ y lo añadimos a Q. De aquí, $\xi \to [b]$ y $\xi \not\to [a]$, luego $[b] \not\to [a]$. □

Por lo anterior, necesitamos como máximo un átomo para cada relación negativa, por lo tanto, $|A(M^*)| \leq |R^-|$.

En este paso del ciclo i-ésimo tocaría forzar las restricciones negativas de traza (Algorithm 1). Intuitivamente uno pensaría en forzar $R_i \cup R_p^{i-1}$, sin embargo, en [10] los autores argumentan que es más eficiente forzar solo las restricciones de traza para R_i y para las relaciones de R_p^{i-1} cuyo arco invertido está presente en G^*, es decir, para toda $a \not\leq b \in R_i$ y para toda $a \not\leq b \in R_p^{i-1}$ tal que $[b] \not\leq [a]$ en G^*. Para todas ellas conseguiríamos $Tr(b) \not\subseteq Tr(a)$. Para las relaciones $a \not\leq b \in R_p^{i-1}$ tal que $[b] \leq [a]$, no haríamos nada. La razón es que al aplicar Algorithm 6 hemos obligado a que se cumplan las relaciones invertidas

de R_i, y no las de R_p^{i-1}, sin embargo, es posible que con los átomos existentes se den algunas de estas relaciones, esto es $[b] \not\leq [a]$, sin haberlas forzado.

Luego continuaríamos con los procedimientos Algorithm 2 para las relaciones positivas de traza y 4 con el Sparse crossing. Volveríamos a medir la calidad de nuestro Modelo AML, y si no es suficiente, calcularíamos los términos de fijación según Algorithm 5 para mantener la información aprendida, eliminaríamos los terminos T implicados en las relaciones R_i y pasaríamos al ciclo $(i+1)$-ésimo.

Este prodecimiento es idóneo para la construcción de un Modelo AML partiendo de un conjunto de datos de entrenamiento R muy grande. Es computacionalmente más eficiente dividir los datos de entrenamiento en participaciones $D = \bigcup_{i=0}^{n} D_i$ y aplicar el aprendizaje por lotes donde en el ciclo i-ésimo forzaríamos las relaciones $R_i \cup R_p^{i-1}$.

4

Resumen: Algoritmo AML paso a paso

En los capítulos anteriores hemos explicado con detalle el algoritmo AML, empezando con la codificación inicial del problema que resuelve embebiéndolo en determinandas estructuras algebraicas y posteriormente viendo los distintos procedimientos de los que consta, cuyo funcionamiento hemos explicado en detalle demostrando todos los resultados en los que estos se apoyan.

En el presente capítulo exponemos un esquema resumido del funcionamiento del algoritmo AML, paso a paso.

4.1. Inicialización del Algoritmo AML

1. **Obtención de datos.** Obtenemos $D \subseteq \Omega$, siendo Ω el conjunto de datos a estudiar y establecemos la propiedad v a predecir en nuevos datos.
2. **Etiquetado de los datos.** Separamos los elementos de $D = D^+ \cup D^-$,

$$D^+ = \{T \in D/\ T \textit{ cumple } v\},\ D^- = \{T \in D/\ T \textit{ no cumple } v\}.$$

3. **Codificación del problema.** Buscamos un conjunto de constantes $C = \{c_1, \ldots, c_k\}$ que "codifiquen" todos los términos, esto es, $\forall T \in \Omega, \exists c_{i1}, \ldots, c_{il} \in C$ tal que $T = \bigodot_{j=1}^{l} c_{ij}$.
4. **Construcción de las relaciones de entrenamiento.** Definimos el conjunto R de las relaciones de entrenamiento, $R = R^+ \cup R^-$, donde

$$R^+ = \{v \leq T_i^+ / T_i^+ \in D^+\},\ R^- = \{v \not\leq T_i^- / T_i - \in D^-\}.$$

5. **Partición de las relaciones de entrenamiento.** Realizamos una partición de los datos D_i y por tanto, de las relaciones de entrenamiento: $R = \bigcup_{i=0}^{m} R_i,\ R_i \cap R_j = \emptyset, \forall i \neq j$, para aplicar el aprendizaje por lotes.

6. **Inicialización de las relaciones de fijación.** Inicializamos $R_p = \emptyset$, que se irá actualizando en los posteriores ciclos la información aprendida acumulada n el Modelo.

7. **Construcción del grafo G.** Construimos el grafo $G = (V, E)$, donde $V = D_0 \cup C(G) \cup A(G)$, siendo D_0 el conjunto de términos mencionados en las relaciones R_0, $C(G)$ las constantes que codifican el problema y $A(G)$ los átomos (inicialmente $A(G) = \{0\}$). Los E se construyen con los axiomas:
   a) Sea $T \in D$, $\{c_1, \ldots, c_n\} \subseteq C(G) : T = \bigodot_{i=1}^{n} c_i \implies c_i \to T,\ 1 \leq i \leq n$;
   b) Sean $T, S \in D$ tal que $T = \bigodot_{i=1}^{n} c_i$ y $S = \bigodot_{j=1}^{m} c'_j$, $\{c'_1, \ldots, c'_m\} \subseteq \{c_1, \ldots, c_n\} \implies T \to S$;
   c) $\forall a \in D \cup C(G),\ 0 \to a$;
   d) $\forall a \in V,\ a \to a$;
   e) $\forall\ a, b, c \in V : a \to b\ \wedge\ b \to c \implies a \to c$
8. **Construcción del semirretículo M.** A partir de la aplicación

$$GL^a : V \to P(A(G))$$
$$v \to GL^a(v) = \{\phi \in A(G)/\phi \to v\},$$

   definimos el semirreticulo $(M, \odot)$, con

$$M := \{GL^a(X)/X \subseteq V\},$$
$$\odot : M \times M \to M$$
$$(x, y) \to x \cup y,$$

   y el orden: $A \leq_M B \iff A \odot B = B \iff A \subseteq B$.
9. **Construcción del grafo invertido G^*.** A partir de la aplicación $[\_] : G \to G^*,\ a \to [a],\ \forall a \in G$, definimos el grafo $G^* = (V^*, E^*)$, donde $V^* = A^* \cup C(G^*) \cup A(G^*)$, siendo $A^* = \{[\phi]/\phi \in A(G)\}$ las imágenes de los átomos de G, $C(G^*) = \{[c]/c \in D \cup C(G)\}$ el conjunto de constantes de G^*, y $A(G^*)$ los átomos de G^* (inicialmente $A(G^*) = \{0^*\}$). Los arcos E^* vienen de:
   a) $\forall\ a, b \in V : a \to b \implies [b] \to [a]$ en V^*;
   b) $\forall a \in C(G^*) \cup A^*,\ 0^* \to a$;
   c) $\forall a \in V^*,\ a \to a$;
   d) $\forall\ a, b, c \in V^* : a \to b\ \wedge\ b \to c \implies a \to c$.
10. **Construcción del semirretículo M^*.** A partir de la aplicación

$$GL^a : V^* \to P(A(G^*))$$
$$c \to GL^a(c) := \{\phi \in A(G^*)/\phi \to c \text{ en } G^*\}$$

    definimos el semirretículo $(M^*, \odot)$, donde

$$M^* = \{GL^a(X)/X \subseteq V^*\},$$
$$\odot : M^* \times M^* \to M^*$$
$$(a, b) \to a \cup b,$$

    y el orden: $A \leq_{M^*} B \iff A \odot B = B \iff A \subseteq B$.

11. **Estructura algebraica final.** Finalmente obtenemos nuestra estructura completa $S = M \cup M^*$ y $G \cup G^*$, donde para todo $a, b \in V \cup V^*$,

$$b \leq c \iff (\forall \phi \in A(G) \cup A(G^*) : \phi \to b \implies \phi \to c) \iff GL^a(b) \subseteq GL^a(c).$$

4.2. Ciclo n-ésimo del aprendizaje por lotes

En el ciclo $n \geq 0$ partimos de la estructura algebraica obtenida en el ciclo $(n-1)$-ésimo (al ciclo 0 entramos con la estructura que acabamos de inicializar) y realizamos el siguiente proceso:

1. **Elección de las relaciones de entrenamiento.** Tomamos las R_n correspondientes a los datos D_n de la partición inicial considerada y las relaciones de fijación R_p^{n-1} calculadas en el ciclo anterior ($R_p^{-1} = \emptyset$).
2. **Adición de nuevos términos a los grafos.** Añadimos a G los términos del conjunto D_n que son mencionados en las relaciones R_n y los siguientes arcos: para todo $T \in D_n : T = \bigodot_{i=1}^{n} c_i$ con $c_i \in C(G)$, añadimos $c_i \to T, \forall i \in \{1, \ldots, n\}$. Asimismo, añadimos las imágenes por [] de estos términos a G^* con los correspondientes arcos invertidos.
3. **Fuerce de las relaciones invertidas.** Para todo $a \leq b \in R_n^+$ añadimos el arco $[b] \to [a]$ en G^*, y para todo $a \not\leq b \in R_n^-$ añadimos un nuevo átomo $\xi \in A(G^*) : \xi \to [b] \wedge \xi \not\to [a]$, para garantizar que $[b] \not\leq [a]$
4. **Comprobación de consistencia.** Comprobamos la consistencia de las relaciones anteriores, en el sentido que $[x] \leq [y] \iff GL^a([x]) \subseteq GL^a([y])$. Si existira alguna relación incosistente y está en R_n concluimos que nuestros datos de entrenamiento son erróneos y, por tanto, debemos conseguir datos nuevos. Si la relación incosistente está en R_p^{n-1}, eliminamos dicha relación de R_p^{n-1} y también su correspondiente "pinning term"de G y G^*.
5. **Algoritmo 6: Reducción de cardinal de G^*.** Aplicamos el Algorithm 6 para reducir el cardinal de G^*, manteniendo solo los átomos necesarios para forzar $[b] \not\leq [a]$, para cada $a \not\leq b \in R_n^-$.
6. **Algoritmo 1: Relaciones negativas de traza.** Para cada $a \not\leq b \in R_n^-$ forzamos $Tr(b) \not\subseteq Tr(a)$. También las forzamos para las relaciones de R_p^{n-1} cuya relación invertida se cumple en G^*, es decir, para $a \not\leq b \in R_p^{n-1}$, si $[b] \not\leq [a]$ entonces forzamos $Tr(b) \not\subseteq Tr(a)$. En este punto, las relaciones de R_n^- y las seleccionadas de R_p^{n-1} ya se cumplen, sólo faltaría forzar las positivas.
7. **Algoritmo 2: Relaciones positivas de traza.** $\forall a \leq b \in R_n^+$ forzamos $Tr(b) \subseteq Tr(a)$.
8. **Algoritmo 4: Reducción de cardinal (Opcional).** Aplicamos Algorithm 4 para reducir el cardinal de $A(G)$.

9. **Algoritmo 3: Sparse Crossing.** $\forall b \leq c \in R^+$ forzamos que $GL^a(b) \subseteq GL^a(c) \implies b \leq c$.
   En este punto ya se cumplen todas las relaciones de R_n y las seleccionadas de R_p^{n-1}.
10. **Comprobación de la calidad del Modelo AML (Opcional).** Comprobamos la calidad y si es tan preciso como deseamos, el algoritmo terminaría aquí, en caso contrario proseguimos con el siguiente paso.
11. **Algoritmo 5: Creación de términos y relaciones de fijación.** Creamos los términos y relaciones de fijación que nos permiten "*recordar*" el aprendizaje hecho en este ciclo n-ésimo y añadimos las nuevas pinning relations al conjunto R_p^n.
12. **Eliminación de términos de los grafos.** Eliminamos de G y G^* los términos implicados en R_n.

Este proceso de aprendizaje por lotes puede repetirse hasta $m+1$ veces, debido a que éste es el número de conjuntos de datos de entrenamiento que tenemos. Si después de $m+1$ ciclos no hemos obtenido la precisión buscada, debemos conseguir más datos para proseguir el aprendizaje.

Una vez hemos obtenido un Modelo AML con la precisión requerida, podemos prescindir de las estructuras G, G^*, M^* ya que sólo son necesarias para el entrenamiento, para realizar predicciones sobre el problema sólo es necesaria la estructura M que nos da la descripción de las constantes en átomos (véase apartado 3.7).

4.3. Predicción a través del Modelo AML

Por último, para usar el Modelo AML para realizar la predicción si un nuevo dato, un término $T \in \Omega$ tiene la propiedad v (barra negra vertical) o no, se seguirán los siguientes pasos:

1. **Codificación del nuevo dato en constantes.** Para todo $T \in \Omega$, aplicando la codificación realizada en 2.2, sabemos que $\exists c_1, \ldots, c_n \in C(G)$ tales que $T = \bigodot_{i=1}^{n} c_i$.
2. **Resultado de la predicción.** Sabemos que T tiene una barra negra vertical sí, y solo si,

$$v \leq T \iff GL^a(v) \subseteq \bigcup_{i=1}^{n} GL^a(c_i).$$

5

A menos átomos, menor error

En capítulos anteriores se ha mencionado que nos interesa reducir al mínimo el número de átomos puesto que esta cantidad está directamente relacionada con la tasa de error. En este capítulo, explicamos cómo en [10] los autores encuentran una cota superior de la tasa de error, que es la que nos muestra esa proporcionalidad.

5.1. Cálculo de la tasa de error del algoritmo AML

Proposición 5.1. *La tasa de error de un modelo atomizado que responde bien R preguntas, para predecir una propiedad P verifica que*

$$\epsilon \lesssim \frac{ZC\ln(2)}{R},$$

donde Z es el número de átomos del modelo, C es el número de constantes y R el cardinal del conjunto de respuestas que sabemos que predice bien.

Por tanto, error esperado es directamente proporcional al número de átomos del modelo.

Demostración. Sea $\mathcal{C}$ el conjunto de posibles modelos para resolver un problema dado. Los modelos que obtenemos mediante el Algoritmo AML son semirretículos que predicen la propiedad v para los elementos de D de manera correcta.

Supongamos entonces que tenemos un modelo con Z átomos. Entonces la probablilidad de que un modelo de este tipo tenga una tasa de error peor que una cierta cantidad ϵ es

$$P(\text{fallo} > \epsilon \mid R \text{ correcto } \wedge Z \text{ átomos}), \tag{5.1}$$

donde "R correcto"denota que predice bien la posesión de la propiedad P para unos ciertos R ejemplos.

Aplicaremos dos fórmulas, por un lado, el *Teorema de Bayes* nos dice que $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$, y por otro lado, la probabilidad condicionada cumple que $P(A|B) = \frac{P(A \cap B)}{P(B)}$. En nuestro caso, la fórmula (5.1) nos queda:

$$P(\text{fallo} > \epsilon | R \text{ correcto } \wedge Z \text{ átomos})$$

$$= \frac{P(R \text{ correcto} \wedge Z \text{ átomos} | \text{fallo} > \epsilon) \cdot P(\text{fallo} > \epsilon)}{P(R \text{ correcto} \wedge Z \text{ átomos})} \tag{5.2}$$

que, por la fórmula de la probabilidad condicionada, queda:

$$P(R \text{ correcto} \wedge Z \text{ átomos} | \text{fallo} > \epsilon) \cdot P(\text{fallo} > \epsilon) = P(R \text{ correcto} \wedge Z \text{ átomos} \wedge \text{fallo} > \epsilon)$$

$$= P(R \text{ correcto} \wedge \text{fallo} > \epsilon | Z \text{ átomos}) P(Z \text{ átomos})$$

que, sustituyéndolo en la fórmula (5.2), nos queda:

$$P(\text{fallo} > \epsilon | R \text{ correcto} \wedge Z \text{ átomos}) = \frac{P(R \text{ correcto} \wedge \text{fallo} > \epsilon | Z \text{átomos}) P(Z \text{ átomos})}{P(R \text{ correcto} \wedge Z \text{ átomos})} \tag{5.3}$$

Existe una gran cantidad de posibles modelos, con cantidades de átomos muy grandes (hasta 2^C, como se indica más adelante). Con nuestro algoritmo, siempre (o casi siempre) vamos a obtener modelos con un número de átomos Z muy inferior a 2^C, y en este rango de valores hacemos la siguiente hipótesis:

$$P(\text{fallo} > \epsilon \wedge R \text{ correcto } | Z \text{ átomos }) \leq P(\text{ fallo } > \epsilon \wedge R \text{ correcto}). \tag{5.4}$$

Esta hipótesis, utilizada como tal por los autores en [10], intuitivamente tiene sentido porque cuantos más átomos tenemos, más maneras tenemos de obtener modelos que funcionen mal. Además, de nuevo por la fórmula de la probabilidad condicionada se tiene que:

$$P(\text{fallo} > \epsilon \wedge R \text{ correcto}) = P(R \text{ correcto}) \cdot P(\text{fallo} > \epsilon | R \text{ correcto})$$

$$\leq P(R \text{ correcto}) \cdot P(\text{fallo} > \epsilon), \tag{5.5}$$

ya que la probabilidad de tener una cierta tasa de error siempre va a ser mayor que la probabilidad de tener esa misma tasa de error sabiendo que el modelo predice con exactitud las R preguntas anteriores.

Y si la tasa de error es ϵ, la probabilidad de acertar las R preguntas es $(1-\epsilon)^R$.

Por tanto, de (5.3), (5.4) y (5.5), nos queda:

$$P(\text{fallo} > \epsilon | R \text{ correcto} \wedge Z \text{ átomos}) \leq \frac{P(\text{fallo} > \epsilon)(1-\epsilon)^R}{P(R \text{ correcto} | Z \text{ átomos})}. \tag{5.6}$$

Si tomamos un ϵ lo bastante bajo, se tiene que $P($ fallo$> \epsilon) \approx 1$, mientras que la probabilidad del denominador se calcularía de la siguiente manera:

$$P(R \text{ correcto}|Z \text{ átomos}) = \frac{|\mathcal{C}_{R\wedge Z}|}{|\mathcal{C}_Z|}$$

donde $\mathcal{C}_Z$ es el conjunto de modelos con Z átomos y $\mathcal{C}_{R\wedge Z}$ el conjunto de modelos con Z átomos que responde bien las preguntas R.

Si denotamos por σ al riesgo que estamos dispuestos a tomar para tener un modelo con una tasa de error peor que ϵ_σ, se tiene que:

$$\sigma = \frac{|\mathcal{C}_Z|(1-\epsilon_\sigma)^R}{|\mathcal{C}_{R\wedge Z}|},$$

y tomando logaritmos, obtenemos:

$$\ln(\sigma) = \ln(|\mathcal{C}_Z|) - \ln(|\mathcal{C}_{R\wedge Z}|) + \ln(1-\epsilon_\sigma)^R.$$

Sabemos que en general $1 < |\mathcal{C}_{R\wedge Z}| << |\mathcal{C}_Z|$, y por tanto,

$$\ln(|\mathcal{C}_Z|) - \ln(|\mathcal{C}_{R\wedge Z}|) \approx \ln(|\mathcal{C}_Z|).$$

De lo anterior se deduce que

$$\ln(\sigma) \approx \ln(|\mathcal{C}_Z|) + R\ln(1-\epsilon_\sigma). \tag{5.7}$$

Vamos a acotar ahora $\ln(|\mathcal{C}_Z|)$, buscando primero una cota de $|\mathcal{C}_Z|$. Consideraremos que dos átomos son iguales si están anexados a las mismas constantes, es decir, si $GU^c(\epsilon_1) = GU^c(\epsilon_2)$, entonces $\epsilon_1 = \epsilon_2$, ya que en la práctica nos dan la misma información. Esto quiere decir, que cada átomo está unívocamente determinado por $GU^c(\epsilon) \subseteq P(C(G))$.

Si tenemos C constantes, entonces el cardinal de $P(C(G))$ es 2^C. Como cada átomo está determinado por un único elemento de este conjunto, podemos acotar el cardinal de $\mathcal{C}_Z$ por $\binom{2^C}{Z}$, s decir, son todas las posibles formas de escoger Z átomos distintos teniendo C constantes. Por tanto,

$$|\mathcal{C}_Z| \leq \binom{2^C}{Z}.$$

Ahora vamos a aproximar $\ln(\binom{2^C}{Z})$ usando un resultado conocido, la *fórmula de Stirling*, y una consecuencia de ella: si n es lo bastante grande, se cumple que $ln(n!) \approx nln(n) - n$. Y además, si $k << n$, entonces, $\binom{n}{k} \approx \frac{n^k}{k!}$. Por tanto,

$$\ln(\binom{2^C}{Z}) \approx \ln(\frac{2^{CZ}}{Z!}) = ZC\ln(2) - \ln(Z!) \approx ZC\ln(2) - Z\ln Z + Z$$

$$= ZC\ln(2) + Z(1 - \ln(Z)) \leq ZC\ln(2) + Z = Z(C\ln(2) + 1).$$

Cuando Z es muy grande podemos aproximar esta cantidad por $ZC\ln(2)$, y por tanto,

$$\ln(|\mathcal{C}_Z|) \leq \ln(\binom{2^C}{Z}) \approx ZC\ln(2). \tag{5.8}$$

Busquemos ahora una cota para $\ln(1 - \epsilon_\sigma)$. Si tomamos ϵ pequeño (tendiendo a 0), se tiene que

$$\lim_{\epsilon \to 0} \frac{\ln(1-\epsilon)}{-\epsilon} = \frac{\ln(1-\epsilon) - \ln(1)}{-\epsilon} = \frac{\partial}{\partial x}\ln(1) = \frac{1}{1} = 1.$$

Por tanto,

$$\ln(1 - \epsilon) \approx -\epsilon, \text{ cuando } \epsilon \to 0. \tag{5.9}$$

Aplicando en (5.7) las aproximaciones calculadas en (5.9), obtenemos que:

$$\ln(\sigma) \approx ZC\ln(2) - R\epsilon_\sigma.$$

Despejando ϵ_σ y usando que $\sigma \leq 1$ (por ser una probabilidad) concluimos que:

$$\epsilon_\sigma \lesssim \frac{1}{R}(ZC\ln(2) - \ln(\sigma)).$$

Para cualquier valor razonable de σ, la cantidad $\ln(\sigma)$ es despreciable. Por ejemplo si tomamos $\sigma = 0,0001$, es decir, aceptamos una probabilidad del $0,01\,\%$ de que el error sea mayor que ϵ_σ, $\ln(\sigma) \approx -9.21$ que para cualquier problema mínimamente complejo, va a ser una cantidad mucho menor que $ZC\ln(2)$, es decir, en la práctica esta cota no depende de σ, así que despreciamos el término $\ln(\sigma)$ y obtenemos que:

$$\epsilon \lesssim \frac{ZC\ln(2)}{R}.$$

Y dado que C y R son constantes, obtenemos que el error esperado va a ser directamente proporcional al número de átomos, Z, del modelo obtenido.

Hemos probado esta fórmula para un modelo elegido aleatoriamente que responda bien unas ciertas R preguntas. En nuestro caso, al finalizar la obtención del modelo AML, éste responde bien unas ciertas R preguntas, que son las relaciones de entrenamiento que hemos usado como *input* del algoritmo. Aunque la elección de este modelo no sea exactamente aleatoria, de acuerdo con [10], la tasa de error de los modelos obtenidos con este algoritmo se ajusta bastante bien a la dada en la demostración.

6

Conclusiones y líneas futuras

El presente capítulo está dedicado a conclusiones generales extraídas tras la realización del presente trabajo, así cómo líneas de investigación a abordar en el futuro.

6.1. Conclusiones

Las principales observaciones que extraemos de la realización del presente Trabajo Fin de Grado son las siguientes:

1. Tras el análisis exhaustivo del algoritmo AML, concluimos que efectivamente presenta mejoras sobre aquellos algoritmos de ML (enfoque estatístico) basados en minimización de funciones, uno de los principales problemas de estos métodos. El algoritmo AML no usa dicho método, y además, no necesita parámetros iniciales Esto evita su estancamiento en la búsqueda de una solución.
2. En el estudio del algoritmo AML nos hemos preguntado cuánto tiempo o nivel de procesamiento se necesita para la construcción del modelo, y si este nuevo métodoo presenta ventajas con los métodos clásicos de ML. Por ello, de forma complementaria a la realización del presente trabajo, y destacado como contribución propia, hemos realizado la programación en Python del algoritmo AML, con la finalidad de compararlo con algoritmos de ML clásico. A fecha de presentación del presente trabajo hemos finalizado dicha programación y la codificación del problema de *reconocimiento de números escritos a mano*. En las próximas semanas haremos un set de pruebas de computación, resolviendo este problema a través del algoritmo AML y de algoritmos ya conocidos de ML/ Deep Learning para obtener así resultados empíricos comparativos sobre su eficiencia.
3. En el estudio del algoritmo AML, hemos encontrado la posibilidad de optimizar el procedimiento de los algoritmos 1 y 3. En el primero de estos, dentro de la función *FindStronglyDiscriminantConstants*, la búsqueda de c

es prescindible. En efecto, siendo a una constante y b un término (negativo) de G, se busca una constante $c \in C(G)$ tal que $c \to a$ y $\exists \zeta \in Tr(b) \subseteq A(G^*)$ cumpliendo $\zeta \not\to [c]$. Sin embargo, por construcción (y evolución) del grafo G, sólo existe un único "nivel" de constantes, por lo que c debe ser siempre igual a a. Dicho de otro modo, si $c \to a$ siendo $c, a \in C(G)$, entonces $c = a$. En el Algorithm 3 proponemos comprobar si las trazas se mantienen en el último bucle, evitando la adición de átomos innecesarios (véase Observación de Algorithm 3).

6.2. Líneas futuras

La realización del presente TFG da lugar a varias líneas interesantes de trabajo:

1. Como trabajo a muy corto plazo, como se mencionó en el apartado de conclusiones, queda por finalizar el set de pruebas aplicándolo a problema de reconocimiento de dígitos escritos a mano, y extraer conclusiones empíricas sobre la eficiencia del algoritmo AML en comparación con otros ya conocidos de ML. Asimismo, se podría extender dicho set de pruebas a otros problemas mencionados en [10], como por ejemplo:
   - *The Queens completion problem*: Dado un tablero de ajedrez nn en el que ya están colocadas algunas reinas, el problema trata de encontrar cómo colocar una reina en cada fila restante de modo que no haya dos reinas que se ataquen entre sí.
   - Resolver sudokus y cubos de Rubik.
   - Encontrar *caminos hamiltonianos*, que son caminos en un grafo de manera que se visitan todos los nodos del grafo una sola vez.
2. Para conseguir una mayor escalabilidad o nivel de aplicación del algoritmo AML, un ámbito de estudio muy interesante sería el de conseguir conseguir automatizar o algunas reglas para codificación de los problemas de clasificación a resolver.
3. Sería asimismo interesante encontrar formas de optimizar el algoritmo de obtención del modelo, puesto que, sobre todo para problemas muy complejos, entendemos que el procesamiento con estructuras de grafos podría requerir de mucho tiempo y de un hardware muy potente para funcionar.
4. También se podría estudiar la aplicación del algoritmo AML a otros tipos de problema, especialmente a problemas reales del sector empresarial. En el presente trabajo sólo hemos estudiado su uso como algoritmo de Clasificación, no obstante, las fuentes estudiadas defienden que puede aplicarse a otros tipos de problema (véase [9]).

Bibliografía

[1] *ALMA Project: Human Centric Algebraic Machine Learning*, funded by European Union's Horizon 2020 research and innovation programme under grant agreement No 952091. https://alma-ai.eu/index.php/publications.

[2] Bishop, C. M. (2006).*Pattern Recognition and Machine Learning*. Springer. ISBN 0-387-31073-8.

[3] Burris, S. & Sankappanavar, H. P. (1981). *A course in universal algebra*. Springer-Verlag.

[4] Davey, D.A & Priestley, H.A. (2002) [1990]. *Introduction to Lattices and Order (2nd. Ed.)*. Cambrigde University Press.

[5] Diestel, R. (2005) [1997, 2000]. *Graph Theory (Electronic Ed.)*. Springer.

[6] Godsil, C. & Royle, G. (2001). *Algebraic Graph Theory (1st Ed.*. Springer.

[7] Kohavi, R. & Provost, F. (1998). *Glossary of terms*. Machine Learning, 30: 271–274. DOL: https://www.researchgate.net/publication/200473546_Glossary_of_Terms.

[8] Mannila, Heikki (1996). *Data mining: machine learning, statistics, and databases*. Int. Conf. Scientific and Statistical Database Management. IEEE Computer Society.

[9] Martin-Maroto, F., Abderrahaman N. & García de Polavieja, G. (2023) *Algebraic Machine Learning: a new program for Symbolic AI*. DOL: https://www.algebraic.ai/aml/. [Fecha de consulta: 22-04-2023].

[10] Martin-Maroto, F. & García de Polavieja, G. (2018). *Algebraic Machine Learning*. DOL: arXiv:1803.05252v2.

[11] Mitchell, T. (1997). *Machine Learning*, McGraw Hill.ISBN 0-07-042807-7.

[12] Russell, S. & Norvig, P. (2003) [1995]. *Artificial Intelligence: A Modern Approach (2nd ed.)*. Prentice Hall. ISBN 978-0137903955.

[13] Martin-Maroto, F. & García de Polavieja, G. (2021). *Finite Atomized semilattices*. DOL: arxiv.org/abs/2102.08050.

[14] Python Software Foundation. *Python*. https://www.python.org/.

[15] *Introduction to Machine Learning. The Wikipedia Guide*. DOL: https://studylib.net/doc/25871992/introduction-to-machine-learning#. [Fecha de consulta: 22-04-2023].

Introduction to Algebraic Machine Learning

Daniel Jaén Guedes
Facultad de Ciencias • Sección de Matemáticas
Universidad de La Laguna
alu0101360676@ull.edu.es

Abstract

We study a recent alternative to classical ML (Machine Learning), which commonly use function minimization. The new method is based on Algebra, using graphs and semilattices to find a model, it is completely parameter-free, and when applied in experiments, no evidence of overfitting has been found [3]. We focus on how to use it to solve a classification problem, that is, to obtain a model that allows us to predict whether an element possesses a certain property.
We make our own proofs to justify the model and we program the Algebraic Machine Learning (AML) algorithm in Python, in order to be able to test how it performs in real cases against statistical ML classification algorithms.

1. Motivation of AML

The creation of this new AML algorithm was motivated by some problems inherent to the classical statistical methods, such as getting stuck on local minima and over-fitting. The main differences are:

Clasical Machine learning	Algebraic Machine learning
Function minimization	Cardinal minimization
Error functions tend to be very complex and it is hard to find minima or to not get stuck on local minima	Does not need to find minima
Depends on a huge number of parameteres which might need tuning with heuristic procedures.	Parameter free
There is risk of overfitting	No evidence of overfitting has been found

2. Construction of the AML Model

In this document we address how to solve *the problem of the vertical bar*, that is, identifying if a given 2x2 image consisting of black and white pixels (*term* T) contains a black vertical bar v, i.e., if $v \leq T$ holds. Before starting the process of obtaining a model, it is required to encode the problem into a set of *constants*.

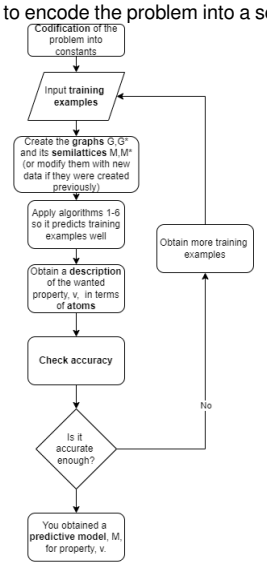

Figure 2: *Flowchart of the AML Method.*

Firstly, the problem is encoded using 8 constants, which are the individual pixels divided by color and position. Thus, we can obtain any term T as the merge of these constants, e.g.:

Figure 3: *Decomposition of the term T_1^+ into 4 constants.*

After applying the AML algorithm (Figure 2) to a given set of terms T (*Training Set*), we obtain the following Algebraic Model:

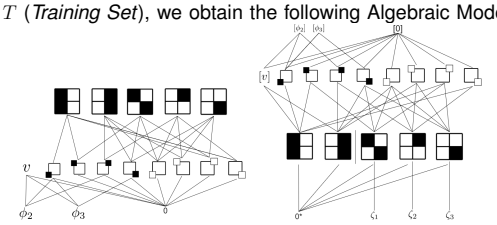

Figure 4: *Grahp G and Inverted Graph G^* (AML output).*

Hence, the property "having a black vertical bar" is determined by the presence of atoms ϕ_2, ϕ_3 edged to v, which means that for a given term T, the AML Model will predict that T has a black vertical bar, or equivalently, $v < T$ if, and only if, the atoms ϕ_2, ϕ_3 of v are also atoms of T (through trasitivity).

For instance, if we consider the term T_1^+ in the previous Figure 3, since both ϕ_2, ϕ_3 are edged to the first constant, and the latter is edged to T_1^+, the AML Model predicts it as positive, that is: "T_1^+ has a black vertical bar".

3. Conclusions

In the study of the AML algorithm, we have found the possibility of optimizing it, specifically in Algorithms 1 and 3, by demonstrating that part of the processing is dispensable.
As a complement to the present work, and as our own contribution, we have carried out the Python programming of the AML algorithm and coded the handwritten number recognition problem. We will perform a set of computational tests, solving this problem using the AML algorithm and well-known ML/Deep Learning algorithms, in order to obtain comparative empirical results on their efficiency.

References

[1] Davey, D.A & Priestley, H.A. (2002) [1990]. *Introduction to Lattices and Order (2nd. Ed.)*. Cambrigde University Press.

[2] Godsil, C. & Royle, G. (2001). *Algebraic Graph Theory (1st Ed.*. Springer.

[3] Martin-Maroto, F. & García de Polavieja, G. (2018). *Algebraic machine learning.* Disponible en: arXiv:1803.05252v2.