Universidad de San Carlos de Guatemala
Escuela de Ciencias Físicas y Matemáticas
Departamento de Física

MODELO DE EVOLUCIÓN DE UN CUESTIONARIO DINÁMICO

APLICACIÓN DE SISTEMAS COMPLEJOS Y MACHINE LEARNING

Jorge Alejandro Rodríguez Aldana

Asesorado por
Lic. Cristian José Álvarez Bran
M.Sc. Edgar Damián Ochoa Hernández

Guatemala, agosto de 2023

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

ESCUELA DE CIENCIAS FÍSICAS Y MATEMÁTICAS

MODELO DE EVOLUCIÓN DE UN CUESTIONARIO DINÁMICO

APLICACIÓN DE SISTEMAS COMPLEJOS Y MACHINE LEARNING

TRABAJO DE GRADUACIÓN
PRESENTADO A LA JEFATURA DEL
DEPARTAMENTO DE FÍSICA
POR

JORGE ALEJANDRO RODRÍGUEZ ALDANA

ASESORADO POR
LIC. CRISTIAN JOSÉ ÁLVAREZ BRAN
M.SC. EDGAR DAMIÁN OCHOA HERNÁNDEZ

AL CONFERÍRSELE EL TÍTULO DE

LICENCIADO EN FÍSICA APLICADA

GUATEMALA, AGOSTO DE 2023

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
ESCUELA DE CIENCIAS FÍSICAS Y MATEMÁTICAS

CONSEJO DIRECTIVO

DIRECTOR	M.Sc. Jorge Marcelo Ixquiac Cabrera
REPRESENTANTE EGRESADO	Lic. Urías Amitaí Guzmán García
REPRESENTANTE DOCENTE	M.A. Pedro Peláez Reyes
REPRESENTANTE DOCENTE	Arqta. Ana Verónica Carrera Vela
REPRESENTANTE ESTUDIANTE	Elvis Enrique Ramírez Mérida
REPRESENTANTE ESTUDIANTE	Oscar Eduardo García Orantes
SECRETARIO	M.Sc. Edgar Damián Ochoa Hernández

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

EXAMINADOR	Ph.D. Juan Adolfo Ponciano Castellanos
EXAMINADOR	Ph.D. Enrique Pazos Ávalos
EXAMINADOR	Lic. José Carlos Alberto Bonilla Aldana

Escuela de Ciencias Físicas y Matemáticas

Ref. D.DTG. 004-2023
Guatemala 26 de septiembre de 2023

El Director de la Escuela de Ciencias Físicas y Matemáticas de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Coordinador de la Licenciatura en Física Aplicada, al trabajo de graduación titulado: **"Modelo de Evolución de un Cuestionario Dinámico Aplicación de Sistemas Complejos y Machine Learning",** presentado por el estudiante universitario Jorge Alejandro Rodríguez Aldana, autoriza la impresión del mismo.

IMPRÍMASE.

"ID Y ENSEÑAD A TODOS"

M.Sc. Jorge Marcelo Ixquiac Cabrera
Director

Ref. 77-2023
JMIC/Paola

AGRADECIMIENTOS

A Dios, por permitirme llegar hasta acá.

A mis padres, quienes con mucho amor me han apoyado en cada paso de mi vida y con su esfuerzo y cariño me han dado muchísimas oportunidades que he sido afortunado de tener. Agradezco su paciencia y confianza, porque sin importar mis decisiones ustedes siempre están allí para mí. No hay acción que pueda realizar que exprese cuán agradecido estoy con ustedes.

A mis asesores, maestros y amigos: Damián y Cristian, quienes desde el principio de la carrera han sido guía y ejemplo para mí. Trabajar con ustedes ha sido un verdadero gusto.

A mis amigos, por hacer de todo este viaje una aventura.

A mis profesores universitarios, por crear una cultura de ciencia y aprendizaje extraordinaria. Gracias a ustedes la escuela es una unidad académica ejemplar, y un espacio para el desarrollo de nuestras ideas.

A Dani, por su cariño, apoyo y motivación constante durante todo este proceso.

Y a todas las personas que han depositado su confianza en mí.

¡Muchas gracias a todos!

DEDICATORIA

Para mis papás, quienes han sido mi guía y ejemplo, solo he podido llegar a donde estoy gracias a ustedes.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

OBJETIVOS

General

Desarrollar un modelo que calcule la evolución temporal de un cuestionario que se adapte a un usuario basado en las respuestas anteriores. Esta solución debe ser escalable y adaptable a cualquier cuestionario.

Específicos

1. Documentar acerca de Sistemas Complejos y Machine Learning.
2. Desarrollar un modelo de evolución de un cuestionario dinámico.
3. Desarrollar una biblioteca de Python para el uso y aplicación del modelo.
4. Realizar un ejemplo de aplicación utilizando el modelo.

INTRODUCCIÓN

En los últimos años, la información ha adquirido una mayor relevancia, las empresas asignan mayores recursos a la infraestructura y los procedimientos de creación, recopilación, proceso y almacenamiento de información, ya que su uso implica una ventaja en la toma de decisiones. En este sentido, mejorar o modernizar las técnicas de recolección de información más utilizadas en investigación, como los cuestionarios y entrevistas, supone un reto importante. Los cuestionarios se diseñan utilizando ítems planteados de forma interrogativa o enunciativa, permitiendo la cuantificación y universalización de la información colectada, su estructura ordenada y concreta permite una interpretación sistemática de los mismos y los hace atractivos para un rediseño en su implementación [2, 5, 17, 30].

Los fuertes avances en la tecnología y la disponibilidad de la información son evidentes, algoritmos y modelos de machine learning se han mejorado al punto de poder crear imágenes, mantener conversaciones e incluso desarrollar código de programación. Empresas como Netflix, Spotify y Google utilizan sistemas de recomendación para sugerir contenido personalizado a sus clientes de manera más eficaz. Incluso herramientas como árboles de decisión ya se emplean para la creación de cuestionarios que evolucionan de acuerdo a respuestas brindadas previamente [36], lo cual es útil cuando existe una alta variedad de temas a evaluar, pero se desea profundizar en los que conciernen más al usuario.

En esta investigación, se propone el diseño y desarrollo de un modelo cuyo objetivo es calcular la evolución de un cuestionario, de manera personalizada para un usuario, a través de un sistema de recomendación. Primero, se exploran los conceptos básicos de la teoría de sistemas complejos y machine learning, necesarios para el planteamiento del modelo. Luego, se expone la definición del modelo y el funcionamiento de este a través de una simulación y análisis teóricos. Se detalla también el proceso de desarrollo del modelo como una biblioteca de Python, se explica su estructura y se documenta su uso. Finalmente se presenta un ejemplo de aplicación del modelo y se realiza una breve discusión de los resultados obtenidos.

1. Sistemas Complejos y Machine Learning

Históricamente, el estudio de la física ha sido principalmente reduccionista, siempre en búsqueda de reducir la complejidad a simplicidad, pero el crecimiento de la misma ha llevado a desarrollar un paradigma opuesto: "La ciencia de la complejidad" [1] [15].

1.1. ¿Qué es un Sistema Complejo?

La complejidad se puede presentar de maneras distintas en escenarios distintos. Todos los fenómenos son en algún grado "complejos", pero que algo sea complejo no lo hace un "Sistema Complejo". Para entender la definición de sistema complejo primero se debe entender el significado de "emergencia", y para hablar de emergencia se debe de hablar sobre las escalas de un sistema.

Cuando un sistema está compuesto por múltiples unidades (agentes) que interactúan entre sí, este sistema puede ser observado por escalas (también llamados niveles o capas). Se puede tener un panorama general compuesto por la totalidad de agentes, pero también se puede detallar cada agente de manera individual, incluso es posible estudiar grupos de agentes que no conformen la totalidad. La *emergencia* se da cuando, al observar múltiples escalas, las acciones e interacciones que sucedan en una generan fenómenos "emergentes" en otra. Entonces, un *sistema complejo* es aquel en el que existe esta emergencia. Este acepta que los procesos que ocurren en las distintas escalas son igual de importantes, los fenómenos que presenta son causados por más que la suma de sus partes y tienen una dependencia no trivial de los agentes que componen al sistema [11, 26, 29, 39, 44].

1.1.1. Discusión sobre los Sistemas Complejos

Debido a la polisemia de la expresión "Sistema Complejo", su uso en la ciencia ha sido controversial. Algunas personas argumentan que el uso deliberado o erróneo de la expresión ha hecho que mucho del trabajo en ciencia de la complejidad

sea pseudo-científico [31]. Pero el debate más marcado habla sobre si puede o no la ciencia de la complejidad ser la respuesta a las limitaciones de los métodos reduccionistas, y si eventualmente, estos serán reemplazados por la misma. La postura que comprende este debate y lo solventa expresa que la ciencia de la complejidad busca estudiar fenómenos y mecanismos específicos, que solo se hacen aparentes bajo la interacción de múltiples agentes, con el objetivo de entender las leyes que los rigen. Se trata de la aplicación y no de la búsqueda de leyes fundamentales [1, 15, 44].

1.1.2. Información e incertidumbre

Se podría pensar que el paradigma de la complejidad está construido sobre conceptos de incertidumbre. Pero en realidad este está construido sobre el concepto de emergencia, la cual se reduce al hallazgo de orden en procesos en los que interactúan múltiples agentes. Esta búsqueda del orden está ligada al uso de la información, la cual cuantifica la dependencia o conexión entre las variables del sistema y el tiempo. Es la información la que inherentemente transporta una cuantificación de incertidumbre, la cual se expresa en términos de entropía.

1.1.2.1. Entropía

En termodinámica, la entropía S se asocia de manera proporcional, al número de microestados Ω que existen en un macroestado particular.

$$S = k \ln(\Omega) . \tag{1.1}$$

Sin embargo, en un sistema físico real, el conteo de microestados supone una tarea imposible, por esto se utiliza la definición de "entropía medida", la cual se define sobre los macroestados y la probabilidad P de que el sistema se encuentre en cada uno de ellos.

$$S = -k \sum_i P_i \ln(P_i) . \tag{1.2}$$

En teoría de la información se utiliza esta misma definición pero con una interpretación distinta, se define la información Q contenida en una afirmación a partir de la probabilidad P de que esta afirmación sea verdadera:

$$Q = -k \log_2(P) . \tag{1.3}$$

Dejando entonces para un conjunto de afirmaciones, la información promedio S como:

$$S = \langle Q \rangle = \sum_i P_i Q_i = -k \sum_i P_i \log_2 (P_i) . \tag{1.4}$$

Dependiendo de la interpretación y uso que se le dé a estas definiciones, la constante k y la base del algoritmo cambian. En el caso de la termodinámica $k = k_b = 1.3807 \times 10^{-23} J/K$ es la constante de Boltzmann y se utiliza el logaritmo natural (base e), esto permite a las ecuaciones tener consistencia con los valores y mediciones físicas. Por otro lado en teoría de la información suele utilizarse $k = 1$ y el logaritmo base 2, de tal manera que la información se mide en bits [3].

1.2. ¿Cómo se estudian los sistemas complejos?

Son varios los mecanismos para el estudio de los sistemas complejos. Uno de los más antiguos y mejor desarrollados es el uso de látices y redes, las cuales ayudan a comprender a los agentes de un sistema y sus interacciones. Por otro lado se encuentran los sistemas dinámicos, estos estudian el comportamiento de los agentes de manera individual utilizando modelos matemáticos y los acoplan para entender sus interacciones. Los sistemas dinámicos se dividen en dos grandes grupos: continuos y discretos, este último estudia la evolución temporal del sistema a través de pasos discretos de tiempo. El uso de machine learning y de herramientas de teoría de la información es otro de los métodos utilizados para el estudio de sistemas complejos. Otros enfoques estudian la adaptación de los sistemas y hacen uso de la teoría de juegos [28, 29].

1.3. Aplicaciones de Sistemas Complejos

Actualmente se realiza investigación en sistemas complejos en múltiples ámbitos académicos. Los orígenes del estudio de los sistemas complejos radican en las ciencias físicas, específicamente en la materia condensada, pero otras áreas de la física como los sistemas hidrodinámicos y la computación cuántica también hacen uso de la teoría de los sistemas complejos. En otras áreas del conocimiento como la biología y química se estudian problemas como el plegamiento de proteínas haciendo uso de sistemas complejos. La biosfera es un ejemplo de un sistema complejo enorme. En sociología, el estudio de las sociedades humanas, en específico la planificación

urbana, la estructura social, las redes sociales y las diferencias entre sociedades. En economía se utilizan los sistemas complejos para el estudio de mercado. Incluso se han presentado contribuciones a la economía desde la física, emergiendo un nuevo sub campo de estudio, la "econofísica" [29].

1.4. ¿Qué es Machine Learning?

Arthur Samuel definió en 1959 el término *Machine Learning* como "el estudio que le da a las computadoras la habilidad de aprender sin ser programadas explícitamente" [9]. Es un sub-dominio de la inteligencia artificial, la cual busca alcanzar metas más complejas [26]. Operativamente se puede definir como los métodos computacionales que utilizan la información del pasado para realizar predicciones o mejorar su rendimiento. Consiste en diseñar algoritmos de predicción eficientes y exactos. El éxito de predicción de estos algoritmos suele depender tanto de la cantidad como de la calidad de los datos, debido a esto, el machine learning está muy ligado a las técnicas de análisis de datos, estadística, probabilidad y optimización [27].

1.4.1. Modelos de Machine Learning

Los algoritmos de predicción de machine learning permiten la construcción de modelos, los cuales son alimentados a partir de datos. Sus parámetros pueden ser ajustados para las tareas deseadas, permitiendo así pasar menos tiempo diseñando los modelos [23]. Existen algunos modelos bien establecidos para resolver algunos tipos de tareas en particular, entre estos se encuentran los clasificadores binarios, regresión, clasificación multiclase, ranking y predicción estructurada. A pesar de tener objetivos particulares, estos modelos son bastante generales y pueden tener diversas aplicaciones. La tabla 1.1 muestra de forma ilustrativa el comportamiento de los mismos.

Se le denomina predictor f a la función que determina una salida a partir de los datos de entrada (llamaremos a estos output e input respectivamente a partir de ahora). El trabajo del aprendizaje automático es generar un predictor óptimo a partir de los datos de entrenamiento. Para distintos modelos, un predictor puede tomar distintos tipos de datos como input y transformarlos en salidas, en la sección 1.4.6 se abordará con mayor detalle el funcionamiento de estos predictores.

Tabla 1.1. Ejemplo de tipos de entrada y salida para distintos modelos. Fuente: tomada de [23]

Modelo	**Entrada**				**Salida**
Clasificador Binario	x	$\longrightarrow$	f	$\longrightarrow$	$y \in \{+1, -1\}$
Regresión	x	$\longrightarrow$	f	$\longrightarrow$	$y \in \mathbb{R}$
Clasificación Multiclase		$\longrightarrow$	f	$\longrightarrow$	*Perro*
Ranking	$(1, 2, 3, 4, 5)$	$\longrightarrow$	f	$\longrightarrow$	$(3, 2, 5, 1, 4)$
Predicción estructurada	*"A tree"*	$\longrightarrow$	f	$\longrightarrow$	*Un árbol*

1.4.2. Vector de características

Un *vector de características* es una representación vectorial de la información contenida en el input. Para generarlo, se deben extraer las características de cada uno de los inputs y asignarles un valor real. Por ejemplo, si el tipo de dato al que pertenecen los inputs es una frase, una característica de esta podría ser la cantidad de caracteres que la componen, y el valor asociado a esta característica para cada input estaría dado por el total de caracteres por los que está integrado, un ejemplo de esto se muestra en la tabla 1.2. A la función que se encarga de extraer las características de cada input se le conoce como *extractor de características*.

Tabla 1.2. Ejemplo de extracción de un vector de características.

"Estoy entre los que piensan que la ciencia tiene una gran belleza." ~ Marie Curie	No. de caracteres: 80 No. de no alfanuméricos: 4 Empieza con mayuscula: 0 Termina con punto: 0	$\longrightarrow$	$\begin{pmatrix} 80 \\ 4 \\ 0 \\ 0 \end{pmatrix}$

Para un input x, llamaremos $\phi(x)$ al vector de características asociado a este:

$$\phi(x) = \begin{pmatrix} \phi^1(x) \\ \phi^2(x) \\ \vdots \\ \phi^d(x) \end{pmatrix}. \tag{1.5}$$

Donde la dimensión d es la cantidad de características asociadas y por tanto $\phi(x) \in \mathbb{R}^d$ [23].

1.4.3. Etiqueta

Se le llama *etiqueta* al output asociado a un input. El objetivo de un modelo es que, una vez entrenado, consiga predecir el valor de dicha etiqueta a partir del input. Las etiquetas no se obtienen solamente de estas predicciones, también pueden estar previamente definidas y ser utilizadas para el entrenamiento del modelo. Para ejecutar un algoritmo de entrenamiento supervisado se necesita un conjunto de datos que contenga tanto los inputs como sus etiquetas asociadas [12].

1.4.4. Training data set

El *training data set* es un conjunto de $n+1$ parejas $\{(\phi_i(x), y_i)\}_{i=0}^{n}$ (vector de características, etiqueta) que se utilizan en el proceso de entrenamiento.

1.4.5. Vector de peso

Para cada característica $\phi^j(x)$ se cuenta con un peso w^j, de tal manera que $W = \left(w^1, w^2, \cdots, w^d\right)$ es el *vector de peso*. Este peso es el que se entrena para que posteriormente sea utilizado para realizar la predicción de la etiqueta. Se le llama "puntaje" a la cantidad que resulta del producto punto entre el vector de peso y el vector de características:

$$\text{score} = W \cdot \phi(x). \tag{1.6}$$

El predictor $f(x)$ hace uso de esta cantidad.

1.4.6. Predictor

Un *predictor* es una función $f(x)$ que toma como entrada un input x y retorna una etiqueta y como salida. El predictor depende del vector de peso W, el cual se mantiene constante una vez realizado el proceso de entrenamiento. A continuación se presentan algunos predictores utilizados en distintos modelos:

1.4.6.1. Predictor de un clasificador lineal

El objetivo de un *clasificador lineal* es clasificar los inputs en dos grupos diferentes, por lo que la etiqueta solo puede tomar dos valores. El predictor $f(x)$ de un clasificador lineal también es llamado "signo" y está dado por la ecuación 1.7 [23].

$$f(x) = \text{sign}(W \cdot \phi(x)) = \begin{cases} 1 & \text{si } W \cdot \phi(x) > 0. \\ -1 & \text{si } W \cdot \phi(x) < 0. \\ \text{indefinido} & \text{si } W \cdot \phi(x) = 0. \end{cases} \tag{1.7}$$

1.4.6.2. Predictor de una regresión

El objetivo de una *regresión* es encontrar una relación entre el input y las etiquetas conocidas en el training data set, realiza las predicciones basada en esta relación. Una regresión lineal indica que la relación es de tipo lineal en cada una de las dimensiones, el predictor de una regresión lineal está dado por:

$$f(x) = W \cdot \phi(x). \tag{1.8}$$

1.4.7. Función de pérdida o "loss function"

La *función de pérdida* $\text{loss}(x, y, W)$ es la que se utiliza para entrenar el vector de peso W. Esta indica el nivel de satisfacción que existe en el uso de W sobre algún vector de características $\phi(x)$ del que se conoce su etiqueta y. A menor pérdida mayor satisfacción. Se presentan algunas funciones de pérdida y los modelos en los que estas son utilizadas:

1.4.7.1. "0-1 loss"

Esta función se utiliza en los modelos de clasificación, sin embargo, no es posible su uso en el proceso de entrenamiento, ya que está compuesta por dos constantes, por lo que la pendiente es cero en todo punto. Más adelante veremos la importancia de la existencia de esta pendiente.

$$\text{loss}_{0-1}(x, y, W) = 1[(W \cdot \phi(x))\, y \leq 0]. \tag{1.9}$$

Donde la función $1[C]$ retorna 1 cuando la condición C se cumple y 0 cuando no. A la cantidad $(W \cdot \phi(x))\, y$ se le conoce como margen, este es positivo cuando los signos de la etiqueta y la predicción coinciden y negativo cuando no. La magnitud del margen indica cuanta confianza se tiene en la predicción realizada [23].

1.4.7.2. "Hinge loss"

Esta función también se utiliza en los modelos de clasificación como una alternativa diferenciable a la función "0-1 loss".

$$\text{loss}_{hinge}(x, y, W) = max(0, 1 - (W \cdot \phi(x))\, y). \tag{1.10}$$

La ventaja de esta función es que sustituye la parte constante distinta de cero, por una recta con pendiente negativa [23].

1.4.7.3. "Squared loss"

Esta función se utiliza comúnmente en los modelos de regresión, está dada en términos del residuo $W \cdot \phi(x) - y$ y encuentra un mínimo cuando la predicción es exactamente igual a la etiqueta. La función está dada por la ecuación 1.11 [27].

$$\text{loss}_{squared}(x, y, W) = (W \cdot \phi(x) - y)^2 . \tag{1.11}$$

1.4.8. Descenso del gradiente

Para entrenar el modelo, se buscan los valores de W que minimizan la función de pérdida para todos los elementos del training data set. Para esto se hace uso de métodos de optimización como el *descenso del gradiente*. Este busca un $W_{trained} \in \mathbb{R}^d$ que minimiza una función objetivo $L(W)$ al desplazar los valores de $W_{trained}$ en dirección opuesta al gradiente $\nabla L(W)$ valuado en $W_{trained}$:

$$W_{trained} = W_{trained} - \eta \nabla L(W)\Big|_{W_{trained}} . \tag{1.12}$$

El coeficiente η representa la tasa o el tamaño del paso que se dará en la dirección de la disminución del gradiente. Su valor puede ser constante o dependiente de la cantidad de iteraciones, esto permite disminuir el tamaño del paso conforme se acerca al mínimo [40]. Existen tres variantes principales del algoritmo de descenso del gradiente en machine learning, estas se describen a continuación:

1.4.8.1. Descenso del gradiente por bloque

Este método utiliza todo el training data set en cada iteración, se hace uso de la ecuación 1.12 iterativamente con una función objetivo $L(W)$ definida por:

$$\nabla L(W) = \frac{1}{N} \sum_{i=0}^{N-1} \nabla \text{loss}(x_i, y_i, W). \tag{1.13}$$

Donde N es la cantidad de elementos en el training data set.

1.4.8.2. Descenso del gradiente estocástico

A diferencia del método anterior, este utiliza únicamente un elemento aleatorio del training data set a cada iteración. Cada N iteraciones (con N el número de elementos en el training data set) se realiza una sucesión con elementos del training data set ordenados de manera aleatoria, luego cada iteración consiste en barrer esta sucesión y aplicar la ecuación 1.12 para el par $(\phi(x_j), y_j)$ en la j-esima posición de la sucesión. En este caso la función objetivo es la función de pérdida:

$$W_{trained} = W_{trained} - \eta \nabla \text{loss}(x_j, y_j, W)\Big|_{W_{trained}}. \tag{1.14}$$

Por lo general este algoritmo suele ser más eficiente, ya que evalúa un solo elemento del training data set (que es usualmente grande pero redundante) a la vez. Sin embargo, presenta fluctuaciones debido a que se ajusta a un único par a la vez, pero estas se ven disminuidas cuando la cantidad total de iteraciones aumenta y el coeficiente η disminuye.

1.4.8.3. Descenso del gradiente por mini-bloques

Este método es una combinación de los dos métodos anteriores. Al igual que en el método estocástico se elige un orden aleatorio para generar una sucesión de elementos del training data set, pero en lugar de barrerlos uno a uno se barren en bloques de n elementos cada uno (mini-bloques) y se aplica el descenso del gradiente por bloque de la sección 1.4.8.1 para uno de estos mini-bloques en cada iteración.

1.5. Aplicaciones de Machine Learning

Existen diversas aplicaciones de las técnicas de machine learning, desde las ciencias físicas pasando por la biología y química, hasta la informática, los negocios y la economía. El desarrollo de algunos conceptos del machine learning ha sido motivado por perspectivas de la física, además, algunas ramas nuevas han surgido como consecuencia de la mezcla entre la física y el machine learning. Algunas de las

aplicaciones en física son en las áreas de física de partículas, cosmología y mecánica cuántica, donde herramientas como la simulación, la regresión y la clasificación son útiles. Una aplicación interesante es la representación de estados cuánticos de sistemas cuánticos de múltiples cuerpos utilizando redes neuronales artificiales [4]. Otra aplicación notable es el uso de machine learning para la predicción en el plegamiento de proteínas, el cual también es estudiado desde los sistemas complejos como se describe en la sección 1.3. También se utilizan las técnicas de machine learning para el análisis predictivo y la toma de decisiones en el ámbito empresarial. El análisis de sistemas complejos es otra de las aplicaciones del machine learning, debido a la capacidad natural de estas técnicas de identificar patrones en datos con una dimensión alta [28], esto se retomará en la sección 1.7. Finalmente, los sistemas de recomendación son un ejemplo del empleo de herramientas de machine learning aplicado a uso comercial a gran escala [38].

1.6. Sistemas de recomendación

Los *sistemas de recomendación* son la respuesta a la problemática que surge cuando existe un catálogo denso de ítems de los cuales un usuario puede escoger, pero se desea mostrarle aquellos con los que esté más alineado, es decir, los ítems más probables de ser elegidos. Estos sistemas tienen aplicaciones en las ventas de productos y servicios, pero no están limitados a estas. Se le llama sistema de recomendación a las herramientas y técnicas de software utilizadas para proveer sugerencias de opciones que tienen una mayor probabilidad de ser de interés para un usuario en particular.

Los sistemas de recomendación regularmente usan herramientas de machine learning para elaborar las recomendaciones. Los datos pueden ser extraídos de forma directa del usuario utilizando evaluaciones o calificaciones de los ítems, pero también pueden ser establecidas a partir de descripciones tanto de los ítems como de los usuarios. Otra forma para obtener información para realizar las recomendaciones es a través de las interacciones de los usuarios, las recomendaciones de Google son un ejemplo de esto [38].

1.7. Sistemas Complejos y Machine Learning

El uso de machine learning para el estudio de sistemas complejos suele darse por la capacidad de estos algoritmos para encontrar patrones en los sistemas. Sin

embargo, existe un debate sobre si el uso de estas herramientas es adecuado para el estudio teórico debido a la no interpretabilidad de sus resultados. A diferencia de los planteamientos deductivos basados en simulaciones físicas, cuyos resultados son inherentemente interpretables, los modelos de machine learning se describen como "cajas negras" que realizan algún razonamiento con los datos y generalizan los patrones que encuentran en ellos, lo difícil es darle una interpretación con significado físico a dicho razonamiento. A pesar de ello, la complejidad de un sistema puede, en ocasiones, introducir una incerteza "extrema" a los modelos físicos, disminuyendo así su predictibilidad y dando paso a la aparición de modelos de machine learning más acertados. Como respuesta a la falta de interpretación de estos modelos, ha surgido el desarrollo de una metodología que utiliza un segundo modelo para interpretar el razonamiento del primero. Esto podría ser engañoso y poco confiable, por lo que una mejor solución es el diseño de modelos de machine learning interpretables, construidos de manera que realicen un razonamiento físico, incorporando así ambas técnicas en el estudio de los sistemas complejos [26, 28, 41].

Pero la relación entre sistemas complejos y machine learning no es en una sola vía. Los modelos de machine learning pueden ser estudiados en sí mismos como sistemas complejos. El uso teórico de los sistemas complejos en el diseño y construcción de modelos de machine learning, en particular, en la optimización de redes neuronales artificiales ha permitido hacer estas más eficientes y menos complejas, reduciendo las conexiones basándose en topologías de sistemas complejos [18].

2. Modelo de evolución de un cuestionario dinámico

2.1. Definiciones previas

2.1.1. Ítem

Se define *ítem* como "cada una de las partes o unidades de que se compone una prueba, un test, un cuestionario" [37]. El uso de este término es común en las Ciencias Sociales, en especial en Psicología. En el contexto de este trabajo, un ítem es una pregunta que puede ser respondida por un usuario.

La preferencia de esta palabra sobre "pregunta" se debe a que el modelo propuesto no se limita a preguntas, sino que puede ser aplicado a cualquier tipo de ítem, como por ejemplo afirmaciones de tipo Likert.

En el contexto de los Sistemas de Recomendación, la palabra "ítem" es utilizada como el término general para denotar qué es lo que un sistema está recomendando a los usuarios [38]. Puesto que en este modelo justamente son los ítems del cuestionario los que serán recomendados, la definición de ítem de sistemas de recomendación coincide con la definición de ítem anteriormente descrita.

2.1.2. Escala de Likert

La *escala de Likert* es una escala de medición de datos de tipo ordinal. Las mediciones ordinales son determinadas por a la posición de un elemento respecto a los demás en una lista ordenada de estos, sin embargo, la medición no establece un valor de distancia entre elementos. Por ejemplo: los puestos de los premios en las carreras son una medición ordinal, puesto que la información que aportan contiene el orden en el que los competidores ingresaron a la meta, pero no las diferencias en las llegadas de los competidores. La escala de Likert comúnmente cuenta con cinco opciones de respuesta: "Totalmente en desacuerdo", "En desacuerdo", "Neutral", "De acuerdo", "Totalmente de acuerdo", de las cuales el sujeto debe seleccionar

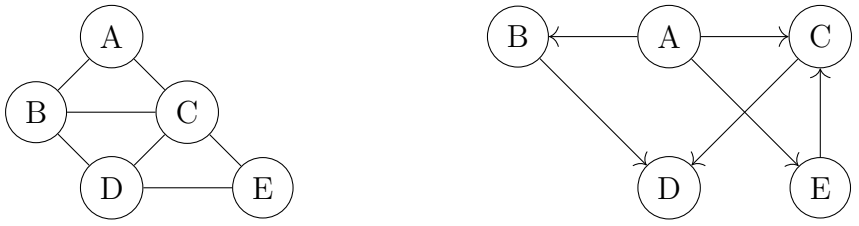
la que sea representativa de su reacción al ítem presentado a modo de afirmación [14, 24]. En esta investigación se le denominará *"ítem tipo Likert"* a un ítem que puede ser respondido mediante una escala de Likert.

2.1.3. Grafo

Un grafo es un objeto matemático, este involucra puntos o "nodos" y conexiones entre estos. Un grafo $G = (N, A)$ consiste de dos conjuntos, el conjunto no vacío de nodos N y el conjunto de aristas A que podría o no ser vacío. El conjunto de aristas contiene subconjuntos de dos elementos del conjunto de nodos que representan una conexión entre estos. Las aristas pueden ser dirigidas, es decir, tener una dirección de la conexión entre un nodo y otro, en este caso las aristas son representadas como listas de dos elementos del conjunto de nodos. A un grafo con aristas dirigidas se le llama grafo dirigido.

Los grafos pueden ser representados mediante diagramas, en estos, los nodos usualmente representados por puntos o círculos, y las aristas por líneas que los conectan. En caso de ser dirigidos, las aristas poseen una flecha indicando la dirección. La figura 2.1a muestra el diagrama del grafo G_a, mientras que la figura 2.1b muestra el diagrama del grafo G_b [13, 42].

$$
\begin{aligned}
G_a =&(\{A, B, C, D, E\}, \\
&\{\{A, B\}, \{A, C\}, \{B, C\}, \{B, D\}, \{C, D\}, \{C, E\}, \{E, D\}\}) \\
G_b =&(\{A, B, C, D, E\}, \\
&\{(A, B), (A, C), (A, E), (B, D), (C, D), (E, C)\})
\end{aligned}
$$



(a) Ejemplo de un grafo. **(b)** Ejemplo de un grafo dirigido.

Figura 2.1. Ejemplos de diagramas de grafos.

2.2. Planteamiento del modelo

Este modelo está inspirado en un sistema de recomendación, donde se busca recomendar ítems tipo Likert a un usuario específico. Se parte de un training data set conformado por ítems respondidos anteriormente por un usuario junto con sus respectivas respuestas, y basado en esto pretende hallar el ítem "más sensato" a realizar a continuación. Esto sucede de manera dinámica, cada vez que el usuario responde un ítem, el training data set cambia y el proceso de recomendación vuelve a ejecutarse. Para la implementación del modelo, a cada ítem se le asocia un vector cuyas coordenadas definen la relación del mismo con una serie de categorías, y a cada respuesta se le asocia una etiqueta que indica la preferencia del usuario por los contenidos del ítem. Luego sobre estos "vectores de características" y "etiquetas" se realiza el proceso de aprendizaje y recomendación. El proceso de aprendizaje consiste en entrenar un "vector de peso" y utilizarlo para predecir las etiquetas de todos los ítems en el cuestionario. El proceso de recomendación se reduce a utilizar estas predicciones para definir una probabilidad de aparición de cada uno de los ítems. El modelo también contempla la posibilidad de que existan varios cuestionarios, y estos puedan relacionarse entre sí. Esto es especialmente útil cuando se desea profundizar en los temas de interés del usuario. Se detallará más sobre esto en la sección 2.2.5.

El modelo desarrollado cumple con las características de un sistema complejo. Este está constituido por agentes bien definidos que operan basados en algoritmos deterministas, pero a otra escala, el comportamiento del sistema tiene un factor aleatorio. Una característica de los sistemas complejos es precisamente que tanto el determinismo como la aleatoriedad juegan un papel importante en su comportamiento general [44].

2.2.1. Definición de categorías

Previo a la elaboración de ítems para un cuestionario, se debe definir la serie de aspectos que se desean evaluar en el mismo, a estos aspectos les llamaremos categorías. Por ejemplo, en un cuestionario en el que se desee evaluar la preferencia por alguna de las estaciones del año, las categorías podrían ser las estaciones: "invierno", "primavera", "verano" y "otoño", este ejemplo se continuará desarrollando a lo largo del capítulo. Se verá más adelante que el orden en el que se enumeren estas categorías es importante.

Una vez definidas las categorías que se evaluarán en el cuestionario, se puede proceder a diseñar y elaborar los ítems. Un ítem puede tener tres tipos de relación

con cada una de las categorías definidas, estas son: “relación directa”, “relación inversa” y “relación nula”. Se dice que un ítem tiene una relación directa con una categoría cuando el enunciado del ítem habla a favor de esta. Cuando el enunciado del ítem habla en contra de una categoría se dice que tiene una relación inversa con la misma. Finalmente, si el enunciado del ítem no habla sobre una categoría se dice que tiene una relación nula con esta. Continuando con el ejemplo de las estaciones del año, un ítem cuyo enunciado diga: “El verano es mucho mejor que el invierno, puedo ir a la playa y tomar el sol” tendría una relación directa con la categoría “verano”, una relación inversa con la categoría “invierno” y una relación nula con las categorías “primavera” y “otoño”. Mientras que un ítem cuyo enunciado diga: “Mis estaciones favoritas son el invierno y el otoño, me encanta el frío” tendría una relación directa con las categorías “invierno” y “otoño”, y una relación nula con las categorías “primavera” y “verano”.

Es aconsejable definir las categorías previo a la elaboración de los ítems, para que estos sean diseñados con intención. Sin embargo, también es posible extraer las categorías que se evalúan en los ítems que componen un cuestionario ya existente.

2.2.2. Vector de características

Dado un cuestionario y sus categorías, se construye un vector de características para cada ítem que se elabore. Este vector contiene la información sobre el tipo de relación que el ítem tiene con cada una de las categorías. Estos vectores viven en el espacio euclideano $\mathbb{R}^d$ cuya dimensión d es igual a la cantidad de categorías a evaluar. Cada categoría representa un grado de libertad en el vector de características, por lo que se debe definir una sucesión de las categorías. El orden de las categorías en esta sucesión se asocia al orden de los ejes del espacio.

Dado un ítem, se construye su vector de características colocando en el n-ésimo eje un valor 1 cuando este tiene una relación directa con la n-ésima categoría en la sucesión, un valor -1 cuando su relación con la n-ésima categoría es inversa y un valor 0 cuando su relación con la n-ésima categoría es nula. En ejemplo sobre las preferencias de las estaciones del año, el vector de características para el ítem “El verano es mucho mejor que el invierno, puedo ir a la playa y tomar el sol” sería: $(-1, 0, 1, 0)$, y para el ítem “Mis estaciones favoritas son el invierno y el otoño, me encanta el frío” el vector de características sería: $(1, 0, 0, 1)$.

2.2.3. Diseño de ítems

Un ítem cuenta con dos partes: el enunciado y las respuestas. El ítem debe ser diseñado para que las respuestas reflejen un valor de preferencia por las categorías correspondientes al ítem. Es por esto que se eligen los ítems de tipo Likert para este modelo, ya que estos reflejan directamente la reacción del usuario ante el enunciado del ítem.

Las respuestas del usuario son tomadas como un valor numérico que representa una etiqueta en el training data set. Las etiquetas asociadas a cada una de las opciones de respuesta de un ítem tipo Likert se exponen en la tabla 2.1.

Tabla 2.1. Etiquetas asociadas a las opciones de respuesta de un ítem tipo Likert.

Respuesta	**Etiqueta asociada**
Totalmente en desacuerdo	-2
En desacuerdo	-1
Neutral	0
De acuerdo	1
Totalmente de acuerdo	2

En algunas implementaciones del modelo, será posible (y recomendable) utilizar una escala de valores continuos en lugar de las opciones de respuesta, dando la alternativa al usuario de seleccionar puntos intermedios entre cada una de estas. Esto agrega precisión a las etiquetas, lo que es útil en el proceso de entrenamiento del modelo. Una forma de implementar esto es hacer uso de "deslizadores" o "sliders" de selección, donde las opciones de respuesta se representen a lo largo de una línea continua, pero el usuario pueda seleccionar cualquier punto de esta. En la figura 2.2 se muestra un ejemplo de este tipo de implementación.

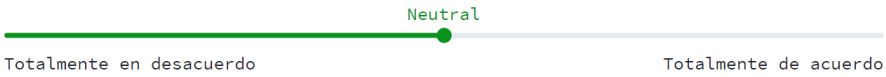


Figura 2.2. Ejemplo de implementación de un deslizador de selección.

2.2.4. Proceso de entrenamiento y recomendación

Ya que el modelo está inspirado en un sistema de recomendación, el objetivo es determinar de alguna manera cuál deberá ser el siguiente ítem a mostrar al usuario.

Para esto se propone un proceso de recomendación que consiste en los siguientes pasos:

1. Agrupar todos los datos del training data set.

2. Entrenar el vector de peso asociado al cuestionario utilizando un algoritmo de regresión. Esto permitirá realizar una predicción de etiquetas para cada ítem en el cuestionario.

3. Utilizar las etiquetas predichas como representación de la preferencia del usuario por un ítem, y calcular la probabilidad de que este sea mostrado a continuación basado en esta y otras variables.

4. Seleccionar el siguiente ítem a mostrar al usuario utilizando la probabilidad de cada uno de ellos. Se debe definir una función que realice esta selección.

Puesto que cada vez que el usuario emite una respuesta un par (enunciado, respuesta) es agregado al training data set, el proceso de entrenamiento vuelve a ejecutarse cada iteración. Esto supone un cambio en las etiquetas predichas en la iteración anterior lo que hace que el sistema evolucione constantemente.

2.2.4.1. Training data set

El training data set se compone de los pares (vector de características, etiqueta) de cada respuesta emitida por el usuario. Es posible que un mismo vector de características mapee a dos o más etiquetas distintas, lo que permite que el modelo considere las inconsistencias que un usuario pueda presentar y que la predicción se ajuste a estas.

2.2.4.2. Método de entrenamiento

Se seleccionó un algoritmo de regresión para este modelo, este realizará una predicción sobre las etiquetas asociadas a cada ítem, permitiendo realizar una recomendación basada en las mismas. Se propone el uso del predictor de regresión lineal[1] expuesto en la ecuación 1.8 y de la función de pérdida de la sección 1.4.7.3 para realizar el entrenamiento.

[1]Originalmente se había planteado utilizar la función "Hinge loss". Sin embargo este planteamiento es limitado porque las predicciones realizadas son binarias, y tomando en cuenta que la clasificación realizada por el mismo está implícita en la ponderación obtenida por el método de regresión, se prefirió este para el desarrollo del modelo.

Se determinó que el uso del algoritmo del descenso del gradiente por bloque expuesto en la sección 1.4.8.1 es la metodología de optimización que mejor se ajusta a este modelo. Esta se prefiere sobre el descenso del gradiente estocástico debido a dos razones, la primera de ellas es la cantidad de muestras en el training data set, y la segunda es la posibilidad de múltiples etiquetas para un mismo vector de características. El descenso del gradiente estocástico brilla cuando la cantidad de muestras en el training data set es elevada, ya que a diferencia del descenso del gradiente por bloque, este no realiza el cálculo sobre todo el training data set en cada iteración [40]. Sin embargo para este modelo, la cantidad de muestras en el training data set está limitada a la cantidad de preguntas respondidas por un mismo usuario, que computacionalmente hablando se mantendrá pequeña. Agregado a esto, el descenso del gradiente estocástico introduce ruido al sacrificar el uso de todo el training data set para minimizar la carga computacional, lo cual es innecesario en esta aplicación. Este ruido se ve magnificado debido a la posibilidad de que un mismo vector de características mapee a distintas etiquetas, ya que en iteraciones distintas el entrenamiento se reajusta a un punto específico y no considera las inconsistencias introducidas por el usuario.

2.2.4.3. Probabilidad del ítem

Se debe asociar una función de peso de probabilidad a cada uno de los ítems (no confundir la función de peso de probabilidad con el vector de peso entrenado). Esta función determinará un valor mayor o igual a cero representativo de la probabilidad de preferencia del usuario por el ítem. La función de peso de probabilidad puede ser definida para cada ítem de forma independiente, lo cual podría ser útil para realizar diseños de cuestionarios con una complejidad alta, sin embargo se aconseja utilizar la misma función para todos los ítems con el objetivo de mantener consistencia en el cálculo de los pesos de probabilidad.

Llamaremos $P(n)$ a la función de peso de probabilidad del n-ésimo ítem. Si esta función está definida para cada ítem en el cuestionario, la probabilidad del n-ésimo ítem $\mathcal{P}(n)$ está dada por:

$$\mathcal{P}(n) = \frac{P(n)}{\sum_{i=0}^{N} P(i)}. \tag{2.1}$$

Donde N representa a la cantidad total de ítems en el cuestionario.

Notemos que la función $\mathcal{P}(n)$ se indetermina cuando $\sum_{i=0}^{N} P(i) = 0$. Esto ocurre solo cuando el peso de probabilidad de todos los ítems es igual a cero (recordemos

que $P(i) \geq 0 \ \forall \ i \in \{0, 1, \cdots, N\}$). En este caso se asigna una probabilidad de $1/N$ a todos los ítems.

En esta investigación se utilizaron dos definiciones de peso de probabilidad: la función de *etiqueta ajustada* P_{AdjLab} dada por la ecuación 2.2 y la función de *etiqueta ajustada con estadística* $P_{AdjStatLab}$ dada por la ecuación 2.3.

Etiqueta ajustada:

Se propone esta función para obtener una probabilidad directamente proporcional al valor de las etiquetas de cada ítem, por lo que cuando el entrenamiento es correcto, los ítems con mayor probabilidad serán aquellos con los que el usuario esté más alineado.

$$P_{AdjLab}(n) = y_n + 2. \tag{2.2}$$

Donde y_n es la etiqueta predicha asociada al n-ésimo ítem. Sumarle un valor de 2 a la etiqueta permite que el peso de probabilidad tenga un valor entre 0 y 4, ya que la etiqueta de un ítem tipo Likert puede tener valores entre -2 y 2.

Etiqueta ajustada con estadística del ítem:

La intención de esta función es retornar un valor proporcional a la etiqueta, pero que tome en cuenta valores estadísticos del ítem. Esto permite agregar o quitar peso a la recomendación basado en los siguientes valores estadísticos: desviación estándar del historial de respuestas del ítem h_n y del historial de respuestas de las categorías del ítem hc_n, y la cantidad de valores en cada uno de ellos respecto al total de respuestas T.

$$\begin{aligned} P_{AdjStatLab}\left(n, \mathrm{std}_w, \mathrm{cstd}_w, \mathrm{cont}_w, \mathrm{ccount}_w\right) = {} & [y_n + 2] \\ & + [\mathrm{STD}(h_n) \cdot \mathrm{std}_w] \\ & + [\mathrm{STD}(hc_n) \cdot \mathrm{cstd}_w] \\ & + \left[\frac{|h_n|}{T} \cdot \mathrm{count}_w\right] \\ & + \left[\frac{|hc_n|}{T} \cdot \mathrm{ccount}_w\right]. \end{aligned} \tag{2.3}$$

Donde $\mathrm{std}_w, \mathrm{cstd}_w, \mathrm{cont}_w, \mathrm{ccount}_w \in \mathbb{R}$ son valores establecidos en la creación del cuestionario, e indican que tan representativas serán las cantidades estadísticas en el cálculo del peso de probabilidad.

En este caso STD(X), $X = \{x_i\}_0^{|X|-1}$ representa la desviación estándar de los valores x_i en el conjunto X:

$$\mathrm{STD}(X) = \sqrt{\frac{1}{|X|}\sum_{i=0}^{|X|-1}\left(x_i - \frac{1}{|X|}\sum_{j=0}^{|X|-1} x_j\right)^2}. \tag{2.4}$$

2.2.4.4. Recomendación basada en la probabilidad

Una vez determinada la probabilidad de cada uno de los ítems, se procede a realizar una recomendación. La función de recomendación $\mathcal{R}(I)$ utiliza el peso de probabilidad de cada ítem en un cuestionario para realizar una selección aleatoria, como se desarrolla en la ecuación 2.5.

$$\mathcal{R}(I) = \mathrm{wrnd}\left((I_0, I_1, \cdots, I_{N-1}), (P(0), P(1), \cdots, P(N-1))\right). \tag{2.5}$$

Donde:

- I_i representa el i-ésimo ítem del conjunto I de N ítems en un cuestionario.
- $P(n)$ es la función de peso de probabilidad del n-ésimo ítem.
- $\mathrm{wrnd}(X, Y)$ es una función que toma como argumentos dos sucesiones X y Y de ítems y probabilidades respectivamente, esta función retorna un elemento $I_r \in X$ elegido de forma aleatoria de una sucesión que contiene el elemento $I_i \in X$ una $P(i) \in Y$ cantidad de veces para cada $i \in \{0, 1, \cdots, N-1\}$.

2.2.5. Múltiples cuestionarios y su interacción

De momento todos los escenarios planteados contemplan un único cuestionario, con un vector de peso entrenado que evoluciona a cada respuesta. Sin embargo esto podría ser limitado en algunos casos donde, por ejemplo, se desee profundizar más en las categorías en las que se muestre un mayor interés. Otro ejemplo podría ser un caso donde existan ítems que no deben ser mostrados al usuario a menos que se cumplan ciertas condiciones. Es por esto que el modelo contempla la creación e interacción de múltiples cuestionarios.

2.2.5.1. Origen y descendencia

La interacción entre cuestionarios se concibe como un grafo dirigido, donde cada cuestionario es representado por un nodo. La dirección de la arista indica el

tipo de relación entre los cuestionarios.

Llamaremos descendencia de A (ofsp (A)) al conjunto de cuestionarios para los que exista una arista que va de A hacia estos. Y llamaremos origen de A (ori (A)) al conjunto de cuestionarios para los que exista una arista que va desde estos hacia A. Por ejemplo: sean A, B, C, D, E cuestionarios representados por el grafo de la figura 2.3, entonces es posible deducir el origen y la descendencia de cada uno de ellos, estos se indican en la tabla 2.2.

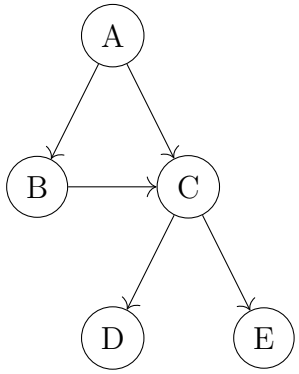


Figura 2.3. Ejemplificación sobre grafo de relaciones.

Tabla 2.2. Ejemplificación sobre deducción de origen y descendencia de cada cuestionario.

Cuestionario	**Origen**	**Descendencia**
A	$\{\}$	$\{B, C\}$
B	$\{A\}$	$\{C\}$
C	$\{A, B\}$	$\{D, E\}$
D	$\{C\}$	$\{\}$
E	$\{C\}$	$\{\}$

2.2.5.2. Vectores de características de un cuestionario

Todo cuestionario cuyo origen sea no vacío, debe tener relación directa o inversa con al menos una categoría que pertenezca a cada uno de los cuestionarios de su origen. Esta relación se define según el contenido del cuestionario, por ejemplo, en el cuestionario de las estaciones del año se podría fabricar un cuestionario descendencia que evalúe categorías: “Lluvia”, “Frío”, “Nieve” y “Viento”, estas surgen como consecuencia del gusto por el invierno, por lo que la categoría a la que pertenece en el origen es el “invierno”. Esto permite construir los distintos vectores de características asociados a cada uno de los cuestionarios del origen utilizando un

procedimiento análogo al de la sección 2.2.2. Continuando con el ejemplo, el vector asociado a este cuestionario sería $(1, 0, 0, 0)$.

2.2.5.3. Etiqueta asociada

La etiqueta asociada a un cuestionario representa la preferencia del usuario por los temas desarrollados en el mismo, se calcula como el promedio de las etiquetas asociadas a cada cuestionario en su origen. Las etiquetas asociadas a cada cuestionario en el origen se determinan calculando el producto punto entre el vector de peso entrenado de cada cuestionario y el vector de características asociado a este. La etiqueta asociada a algún cuestionario A está dada por:

$$y_A = \frac{1}{|\text{ori}(A)|} \sum_{i=0}^{|\text{ori}(A)|-1} W_i \cdot x_{Ai}. \tag{2.6}$$

Donde W_i es el vector de peso entrenado del i-esimo cuestionario en el origen, y x_{Ai} el vector de características de A asociado al i-esimo cuestionario en el origen.

2.2.5.4. Condición de activación

Cuando se trabaja con cuestionarios múltiples, a cada uno se le debe asociar una "condición de activación", esta determina si el cuestionario cumple con los criterios deseados para ser mostrado al usuario. Esta condición se puede representar mediante una función que retorna 1 cuando la condición se satisface y 0 cuando no. Suele depender de la etiqueta asociada al cuestionario. Se proponen las siguientes funciones de condición:

Función True

Esta retorna un valor de 1 siempre, es decir, el cuestionario siempre cumple con los criterios para ser mostrado al usuario. Esta función se debe utilizar en el cuestionario de partida como se describe en la sección 2.2.5.6.

$$\text{True}() = 1. \tag{2.7}$$

Función threshold

Esta función retorna 1 cuando la etiqueta y_S del cuestionario S en cuestión alcanza o sobrepasa un valor v determinado (threshold).

$$\text{Threshold}_v(y_S) = \begin{cases} 0 & \text{si } y_S < v. \\ 1 & \text{si } y_S \leq v. \end{cases} \tag{2.8}$$

2.2.5.5. Probabilidad del cuestionario

Al cuestionario también se le debe asociar una función de peso de probabilidad de manera similar al procedimiento realizado en la sección 2.2.4.3. Esta función determinará un valor mayor o igual a cero representativo de la probabilidad de preferencia del usuario por los contenidos del cuestionario. Se recomienda utilizar la función "etiqueta ajustada" definida en la ecuación 2.2 utilizando la etiqueta y_S del cuestionario.

2.2.5.6. Interacción de los cuestionarios

El algoritmo de interacción determina de qué cuestionario se seleccionará el próximo ítem a mostrarse al usuario, para esto se genera un conjunto con los cuestionarios de los que podría realizar esta selección, a este le llamaremos conjunto de selección.

Se debe definir un cuestionario de partida, este siempre debe cumplir la condición de activación. El algoritmo realiza lo siguiente: revisa si la condición de activación se cumple, en cuyo caso añade el cuestionario al conjunto de selección y realiza el mismo procedimiento para la descendencia de forma recursiva. En caso de no cumplirse la condición de activación este cuestionario no es agregado al conjunto de selección y su descendencia no es tomada en cuenta. El procedimiento se detiene cuando no existen más descendencias o ninguno de los cuestionarios cumple la condición de activación.

Debido a la naturaleza recursiva de este algoritmo, se debe imponer la condición que en el grafo de relación entre cuestionarios no debe existir ningún circuito dirigido, de lo contrario el algoritmo podría nunca detenerse si todos los nodos en dicho circuito satisfacen la condición de activación.

2.3. Escenarios teóricos

Para corroborar el funcionamiento de la metodología planteada, se realizaron algunas pruebas teóricas que consistieron en simular la interacción entre un usuario y un cuestionario hipotético. Los usuarios fueron programados para presentar cierta

preferencia ante los ítems del cuestionario. Estas pruebas fueron realizadas haciendo uso de las herramientas desarrolladas en el capítulo 3.

2.3.1. Diseño de las pruebas

Cada cuestionario cuenta con una cantidad de 3^D ítems, correspondientes a todas las permutaciones posibles para fabricar un vector de características de D dimensiones con valores $-1, 0, 1$. Cada una de estas dimensiones representa una categoría a evaluar. La probabilidad de estos ítems se definió utilizando el procedimiento de "etiqueta ajustada" de la sección 2.2.4.3. Se define un usuario para contestar la prueba, a este se le asocia un vector que llamaremos "vector de preferencia" que representa el nivel de preferencia de dicho usuario por cada una de las D categorías del cuestionario, las coordenadas del vector se establecen de manera aleatoria con valores entre -2 y 2, que indican total disgusto y total gusto por la categoría respectivamente. Se realiza la evolución del cuestionario según el proceso de evolución desarrollado en la sección 2.2.4. Conforme los ítems se van presentando al usuario, este calcula la etiqueta de los mismos mediante el producto punto entre el vector de preferencia y el vector de características del ítem. Posteriormente calcula un valor aleatorio distribuido alrededor de la etiqueta calculada utilizando una función que genera números aleatorios siguiendo una distribución normal con una desviación estándar igual a 1. El usuario contesta el ítem truncando este valor aleatorio a alguna de las opciones de respuesta definidas en la tabla 2.1. El proceso se repite n veces, que representa la cantidad de ítems que contesta el usuario.

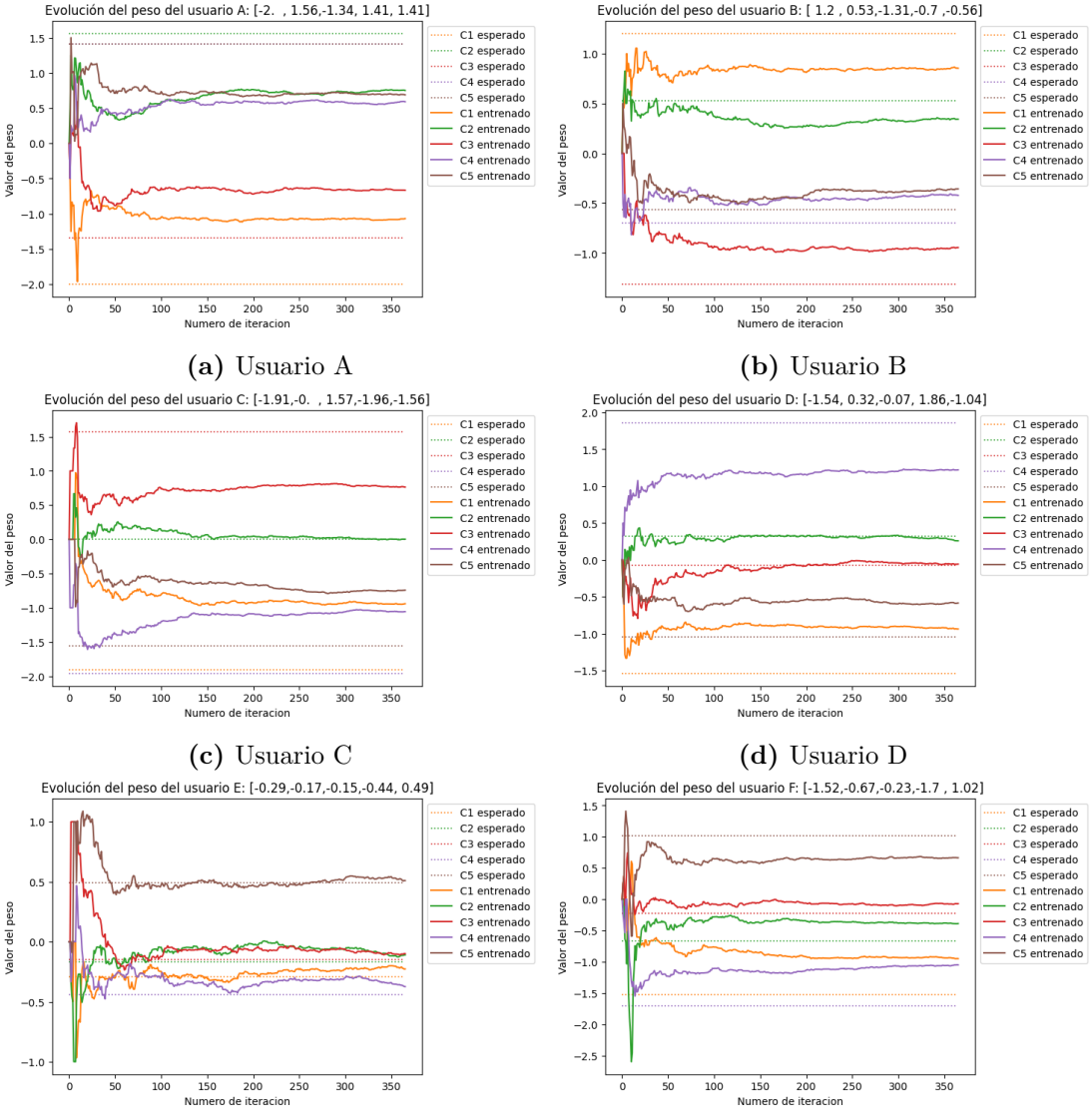
2.3.2. Ejecución básica

Para esta ejecución la dimensión del cuestionario fue definida igual a 5, por lo que se generaron $3^5 = 243$ ítems. Se definieron 6 usuarios (denotados A,B,C,D,E y F) cuyos vectores de preferencia se muestran en la tabla 2.3. Cada usuario contestó un total de 365 ítems, simulando un caso en el que se contesta un ítem al día durante un año. La figura 2.4 muestra la evolución de cada una de las coordenadas del vector de peso para cada usuario, además de los valores esperados dados por el vector de preferencia de cada uno. Se observa que los valores del peso entrenado se estabilizan después de los primeros 50 ítems contestados, pero ocurre un fenómeno, aunque el orden y separación de los mismos parece ser el correcto, los valores parecen estar multiplicados por un factor que los acerca a cero. Este cambio de escala tiene una explicación, y es que al limitar las respuestas a un rango definido, todas las

etiquetas que se salen de este rango serán siempre seleccionadas con el valor extremo. Por ejemplo, un usuario cuyo vector de preferencia sea $(1, 1, 1, 1, 1)$, que pretende contestar un ítem con vector de características $(1, 1, 1, 1, 1)$ realiza un cálculo de etiqueta que resulta en 5, pero al truncar este valor a las opciones de respuesta, el usuario contestará 2. Esto ocurre con todas las etiquetas que sean mayores a 2, por lo que el valor esperado de las etiquetas se ve reducido. Sin embargo la predicción de orden y separación de los valores es correcta. En la sección 2.3.3 se realiza una comparativa de este resultado con el mismo experimento, pero sin truncar las etiquetas.

Tabla 2.3. Vectores de preferencia de los usuarios

Usuario	Vector de preferencia
A	$(-2.00, 1.56, 1.34, 1.41, 1.41)$
B	$(1.20, 0.53, -1.31, -0.70, -0.56)$
C	$(-1.91, -0.00, 1.57, -1.96, -1.56)$
D	$(-1.54, 0.32, -0.07, 1.86, -1.04)$
E	$(-0.29, -0.17, -0.15, -0.44, 0.49)$
F	$(-1.52, -0.67, -0.23, -1.70, 1.02)$



(a) Usuario A **(b)** Usuario B

(c) Usuario C **(d)** Usuario D

(e) Usuario E **(f)** Usuario F

Figura 2.4. Evolución de las coordenadas del vector de peso para cada usuario.

En cada iteración se realizó el cálculo de la entropía del cuestionario utilizando la ecuación 1.2, con P_i igual a la probabilidad del i-ésimo ítem y $k = 1$.

La entropía podría interpretarse como una medida de incertidumbre en la recomendación realizada. Si la distribución de probabilidad de los ítems se asemeja mucho a una uniforme entonces el modelo no tiene una recomendación bien definida. Suponiendo que el modelo sí está bien entrenado, esto implica que si la entropía es alta, el usuario tiene en general igual gusto o aversión por todas las categorías presentadas. Más aún, debido a que el modelo se adecúa a las preferencias del usuario, si estas se mantienen constantes la entropía tenderá a ser constante a través del

tiempo y alternativamente, si la entropía presenta un comportamiento oscilatorio en el tiempo, puede interpretarse como que el usuario no es consistente con sus preferencias.

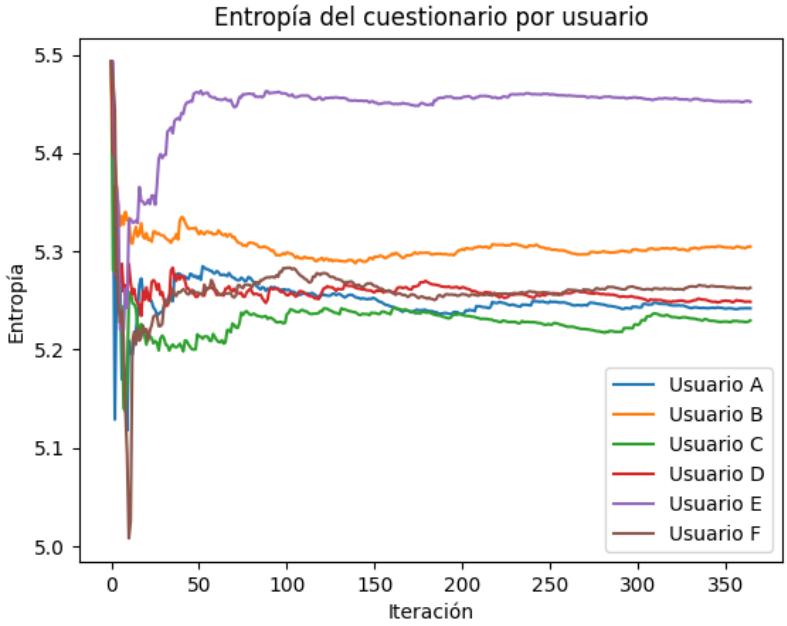
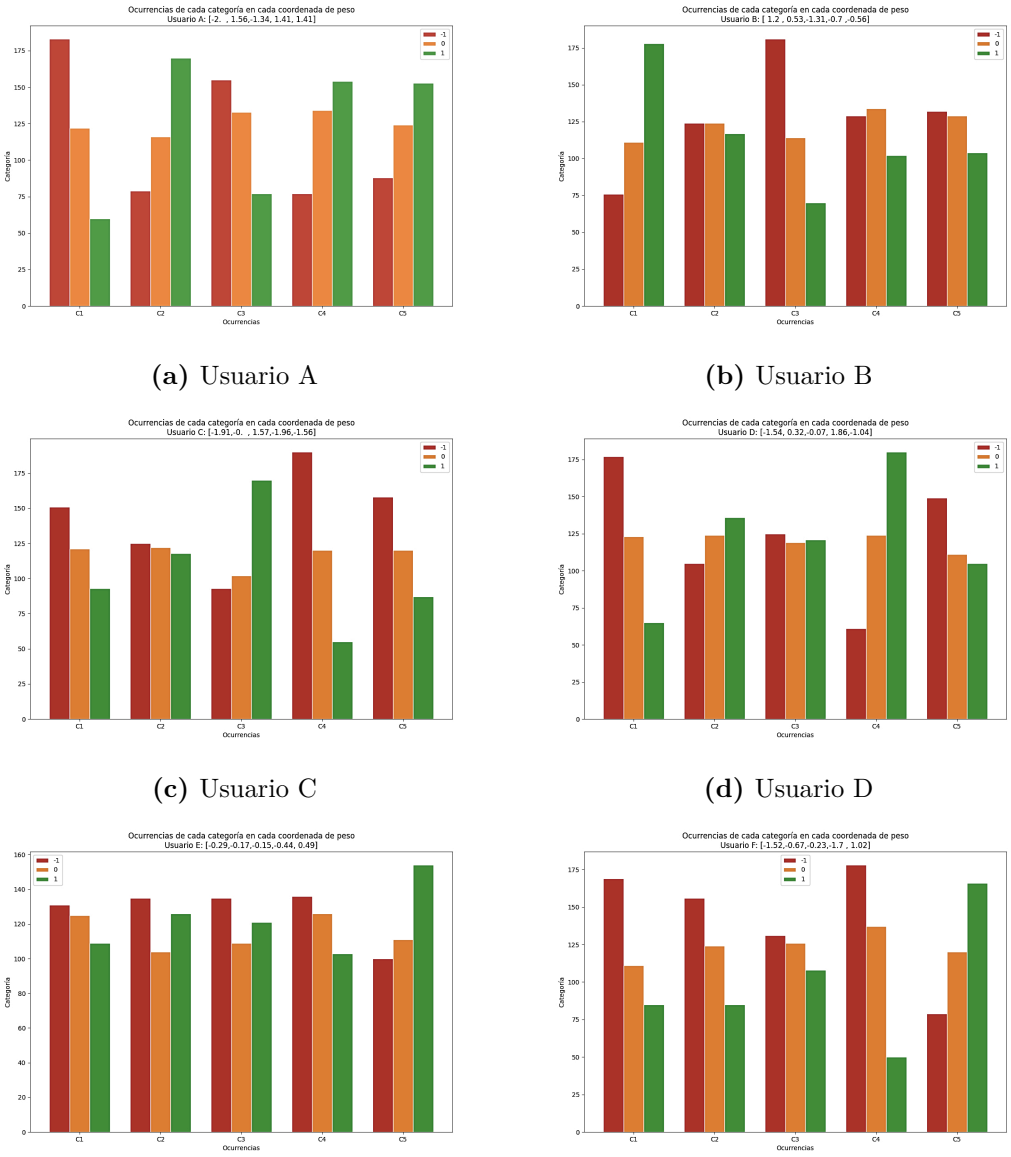


Figura 2.5. Evolución de la entropía del sistema para cada usuario.

Puesto que uno de los factores más importantes del modelo es que las preguntas se ajusten a los usuarios, recomendando aquellas que se alineen con sus preferencias, también se realizó una gráfica por usuario que muestra la cantidad de ítems con relación directa, inversa y neutra para cada una de las categorías del cuestionario. La figura 2.6 muestra estas gráficas. Como es de esperarse, se obtienen más ítems con relación directa a las categorías que tienen un valor positivo y más ítems con relación inversa a las categorías que tienen un valor negativo de peso en el vector de preferencias del usuario. La magnitud de estos valores también afecta de manera proporcional a que tan marcada es la diferencia entre la cantidad de ítems con relación directa e inversa.



(a) Usuario A **(b)** Usuario B

(c) Usuario C **(d)** Usuario D

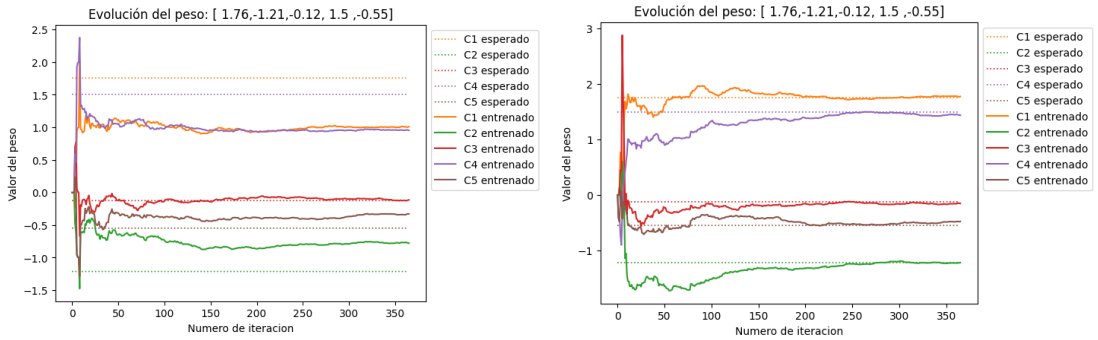
(e) Usuario E **(f)** Usuario F

Figura 2.6. Cantidad de ítems con relación directa, inversa y neutra para cada categoría del cuestionario.

2.3.3. Comparación con el caso no truncado

Como punto de referencia, también se realizó un análisis comparativo entre dos casos, el primero tiene un desarrollo idéntico a la ejecución de la sección anterior y el segundo se realiza dejando la libertad al usuario de seleccionar cualquier valor real como respuesta, es decir, sin truncar los valores a las opciones de respuesta. El vector de preferencia generado aleatoriamente fue $(1.76, -1.21, -0.12, 1.50, -0.55)$.

La figura 2.7 muestra la evolución del vector de peso para cada caso.



(a) Caso truncado

(b) Caso no truncado

Figura 2.7. Comparación de la evolución del vector de peso para el caso truncado y no truncado.

El coeficiente de determinación es uno de los indicadores más utilizados para medir la calidad de un modelo de regresión, este se define en la ecuación 2.9, y se interpreta como la proporción de la varianza de la etiqueta (variable dependiente) que es predecible a partir del vector de características (variable independiente), por lo que valores cercanos a 1 indican una mejor predictibilidad [7].

$$R^2 = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \bar{y})^2}. \tag{2.9}$$

Donde y_i son las etiquetas del training data set, $\hat{y}_i$ son las etiquetas predichas por el modelo y $\bar{y}$ es el promedio de las etiquetas del training data set.

En la figura 2.8 se muestra la evolución del coeficiente de determinación para cada caso. Se puede observar que, aunque el caso no truncado tiene un mejor desempeño que el caso truncado, la diferencia entre estos no es muy grande. Agregado a esto, por lo descrito en la sección 2.1.2, la escala de Likert es de tipo ordinal, lo que indica que los valores puntuales no aportan información más allá del orden de preferencia, por lo que un cambio de escala en la predicción no afecta la calidad de la misma.

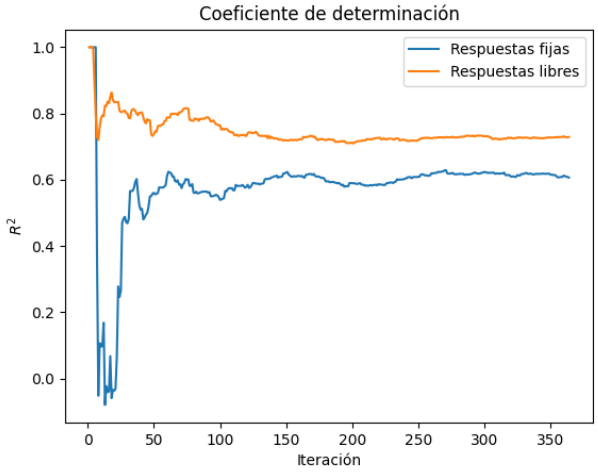
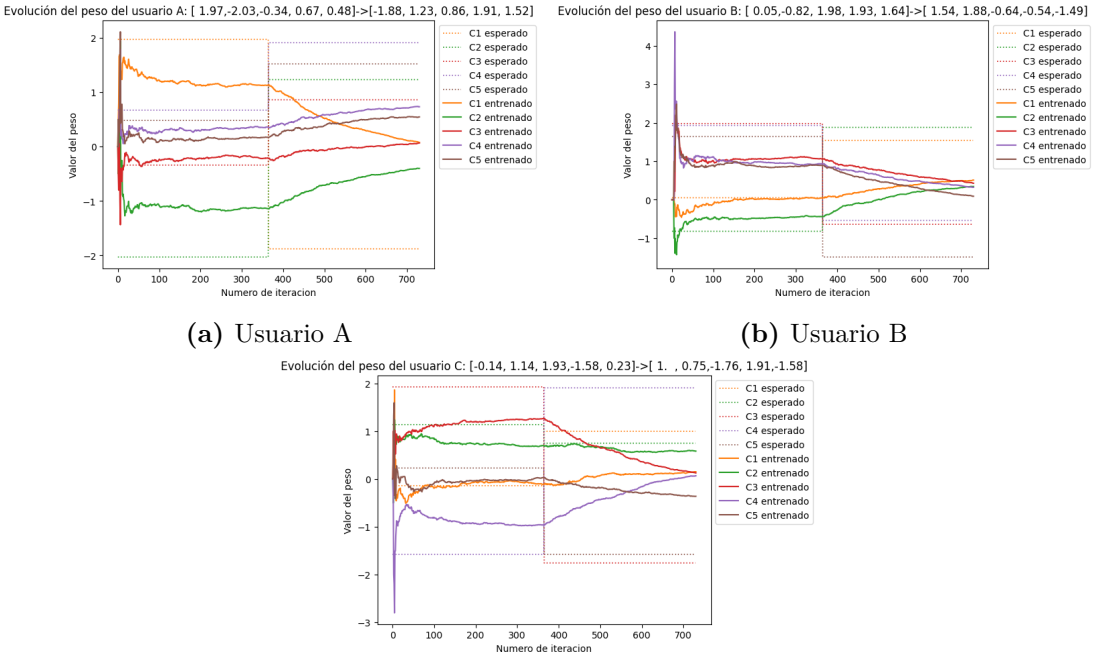


Figura 2.8. Evolución del coeficiente de determinación para el caso truncado y no truncado.

2.3.4. Ejecución con cambio de preferencia

Se realizó una segunda ejecución que modelaba un cambio de preferencias de los usuarios, para esta se siguió el mismo procedimiento que en la sección 2.3.2, pero esta vez se realizaron 730 iteraciones (simulando dos años contestando un ítem cada día) con un cambio en el vector de preferencias en la iteración 360. Esto se realizó para 3 usuarios denominados (A, B y C). La evolución de las coordenadas del vector de peso se muestra en la figura 2.9. Además se tienen las gráficas de la evolución de la entropía y el coeficiente de determinación de cada usuario en las figuras 2.10a y 2.10b respectivamente.

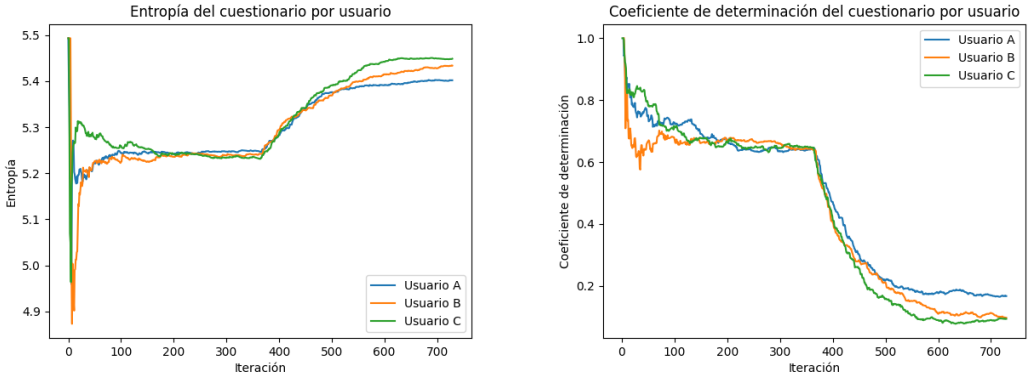
Se observa cómo tanto la entropía como el coeficiente de determinación sufren un cambio brusco en la iteración 360 debido al cambio de preferencia. En futuras investigaciones se propone el uso de estas métricas para detectar cambios en las preferencias de los usuarios y de esta manera conseguir una respuesta más rápida de adaptación en el proceso de entrenamiento.



(a) Usuario A **(b)** Usuario B

(c) Usuario C

Figura 2.9. Evolución de las coordenadas del vector de peso con un cambio de preferencia.



(a) Entropía **(b)** Coeficiente de determinación

Figura 2.10. Evolución de la entropía y el coeficiente de determinación con un cambio de preferencia.

3. Biblioteca de Python

Para hablar sobre una biblioteca de Python, primero se deben abordar otros temas tales como: ¿Qué es un lenguaje de programación?, ¿Qué es Python? o ¿Cuál es la estructura de un programa realizado con Python?

3.1. Lenguajes de programación

Los *lenguajes de programación* se encargan de trasladar información a una computadora para que ejecute tareas y procesos. Esto permite la producción y el desarrollo de software para la creación de programas [20].

Se requiere un mecanismo para que un humano comunique estas instrucciones a una máquina, dependiendo de la profundidad del lenguaje de programación se le denomina un nivel del lenguaje. Estos niveles se pueden categorizar tres tipos:

3.1.1. Lenguaje máquina

Este es el nivel más básico de lenguaje de programación, consiste en una serie de dígitos binarios o *bits* que pueden ser entendidos y procesados directamente por la computadora sin la necesidad de una traducción.

Los bits representan la presencia o ausencia de señales eléctricas, correspondiendo a los dígitos binarios 0 y 1, los valores de apagado y encendido respectivamente.

3.1.2. Lenguaje Ensamblador

El *lenguaje ensamblador* lenguaje es un intermediario entre el lenguaje máquina y el programador. Este utiliza mnemotécnicos que representan una instrucción individual y se asocian al código que ejecuta esta. Estos mnemotécnicos permiten al programador utilizar palabras para indicar una serie de instrucciones.

El lenguaje ensamblador no es portable, es decir, no es posible trasladar un mismo código escrito en lenguaje ensamblador de una computadora a otra, ya que

los mnemotécnicos y sus instrucciones asociadas son dependientes del hardware.

3.1.3. Lenguaje de alto nivel

Un *lenguaje de alto nivel* es un lenguaje de programación que contiene un nivel de abstracción más alto. Este se asemeja más al lenguaje humano o a la notación matemática [20].

Existen diversos lenguajes de alto nivel, ya que estos permiten a los desarrolladores una mejor comprensión y lectura de la lógica detrás del código. Sin embargo, para que las computadoras puedan ejecutar este código debe existir un puente entre el mismo y la computadora.

Este proceso se realiza utilizando un compilador o intérprete, que traduce el código del lenguaje de alto nivel a lenguaje máquina [20]. Según sea el caso, se dice que el lenguaje es compilado o interpretado respectivamente.

3.1.3.1. Lenguajes compilados

Un *lenguaje compilado* realiza una traducción del código a lenguaje máquina para una computadora específica. Es decir, genera una serie de instrucciones en binario diseñadas para un equipo, por lo que no pueden ser trasladadas a otra computadora de manera directa, sino se requiere realizar traducciones distintas para cada uno de los equipos.

Gracias a la manera de operar de los lenguajes compilados, una vez traducidos, estos pueden ser ejecutados de manera más efectiva y rápida.

3.1.3.2. Lenguajes interpretados

Un *lenguaje interpretado* traduce el código a lenguaje máquina una línea a la vez [20]. Este código es ejecutado directamente en el equipo y por lo tanto puede trasladarse de un equipo a otro mientras ambos equipos cuenten con el intérprete.

Los lenguajes interpretados son, por naturaleza, menos eficientes que los lenguajes compilados. Ya que el intérprete debe leer, traducir y ejecutar una línea de código a la vez. Y cada vez que el código vuelve a ejecutarse, el proceso de traducción debe volver a realizarse.

3.1.4. Programación orientada a objetos

La *programación orientada a objetos* es un paradigma de programación, este simplifica la estructura de un programa al utilizar bloques de código reutilizables llamados clases. Las clases son una estructura abstracta y general que se utiliza como base para crear objetos específicos llamados instancias. Las clases definen atributos que caracterizan el objeto, pero no sus valores, ya que estos son asignados a cada instancia de la clase. Las clases también cuentan con métodos, los métodos son funciones asociadas a la clase, estas realizan procedimientos inherentemente relacionados con la clase en sí [8]. La creación de una instancia se realiza a través de un método, a este se le conoce como constructor. Por ejemplo: se podría definir una clase sobre "guitarras". Algunos atributos de esta podrían ser el "color", "tipo de cuerda", "tipo de guitarra" y "marca". Una instancia de esta clase podría ser una guitarra Fender, eléctrica amarilla y con cuerdas de metal, y esta se crearía utilizando el método constructor de la clase "guitarra" que asignaría los valores de los atributos marca, tipo, color y tipo de cuerda a Fender, eléctrica, amarilla y metal respectivamente. Un ejemplo de método para esta clase podría ser "cambiar cuerdas", este cambiaría el valor del atributo tipo de cuerda en la instancia que sea ejecutado.

3.2. ¿Qué es Python?

Python es un lenguaje de programación de alto nivel, interpretado y orientado a objetos [43]. Su sintaxis simple lo hace fácil de leer, lo que permite un desarrollo más ordenado. También soporta módulos y paquetes, lo que alienta a la programación modular y al reúso de código.

Python cuenta con diversas bibliotecas que ayudan a los desarrolladores en la creación de código. Gracias a esto, Python es uno de los lenguajes preferidos para el desarrollo y la investigación en Machine Learning.

Un programa realizado en Python posee una estructura, esta se compone de módulos, los cuales contienen declaraciones formadas de expresiones. Esto se encarga de crear y procesar objetos, dando lugar al funcionamiento del programa [25].

3.2.1. Objetos

En Python los datos que deben ser almacenados son llamados *objetos*. Un objeto es un bloque de memoria, con valores y conjuntos de operaciones asociadas.

Python cuenta con algunos objetos generales predefinidos. Estos tienen la ventaja de ser estándar y son parte intrínseca del lenguaje, lo que los hace eficientes. Dentro de estos objetos predefinidos se encuentran: las palabras `str`; los enteros `int`; los decimales `float`; listas `list`; diccionarios `dict`; tuplas `tuple`; entre otros.

3.2.2. Declaraciones

Un programa en Python se compone principalmente de declaraciones y expresiones. Las expresiones están contenidas en las declaraciones. Las *declaraciones* son las encargadas de la lógica detrás del código, también pueden ser utilizadas para crear otro tipo de objetos o definir operaciones para los mismos.

Algunas declaraciones están programadas por defecto en el intérprete de Python: Las declaraciones de asignación, las cuales crean referencias entre objetos y datos; las declaraciones de selección, las cuales se encargan de seleccionar los casos en los que determinada condición se cumple; las declaraciones de iteración, estas generan una serie de acciones basadas en una secuencia dada; las declaraciones de bucle, las cuales repiten una acción hasta que cierta condición se establezca; las declaraciones de función y método, las cuales se encargan de definir operaciones nuevas; las declaraciones de clase, las cuales se encargan de definir clases nuevas; entre otras declaraciones. Algunos ejemplos se desarrollan en la tabla 3.1.

Tabla 3.1. Tabla de ejemplos de declaraciones usando Python.

Declaración	Ejemplo
Asignación	`a = 'hello'`
Selección	`if a != 'hello'`
Iteración	`for i in range(5)`
Bucle	`while a == 'hello'`

3.2.3. Módulos

En Python, un *módulo* hace referencia a un archivo, que contiene código. Este podría ser programado en otro lenguaje como C, Java o C# y ser llamado como una extensión a Python, o ser programado directamente en Python.

El uso de módulos en Python es lo que hace de este un lenguaje tan popular, ya que incentiva el reúso de código y crea un entorno adecuado para la distribución del mismo.

Un solo archivo de código en Python puede ser un programa en sí mismo. Sin embargo, utilizar módulos para la creación de programas permite una mejor organización además de una fácil reutilización.

3.2.4. Paquetes

Un *paquete* es un directorio que contiene uno o varios módulos de Python. También puede incluir subdirectorios que contengan módulos en sí mismos, a estos se les conoce como subpaquetes.

Comúnmente los módulos contenidos en un paquete suelen estar relacionados entre sí, organizando distintas funciones al rededor de un tema en específico. Aunque este no debe ser estrictamente el escenario.

Un paquete puede ser un módulo en sí mismo. Un directorio puede actuar como un archivo de código de Python sí en él se encuentra un archivo llamado `__init__.py`. Cuando el directorio sea importado como un módulo, el código dentro de `__init__.py` será ejecutado. Normalmente dentro de un archivo `__init__.py` únicamente se realizan importaciones a los módulos contenidos en el paquete, de este modo estos pueden ser importados a través de una sola declaración como submódulos del paquete.

3.2.5. Importación de módulos y paquetes

Los módulos pueden ser accedidos entre sí, es decir, un archivo de código de Python puede acceder al código, lógica y funcionalidad de otro archivo de código. A este proceso se le conoce como *importación* de código, y se realiza utilizando la declaración `import`.

Cuando un módulo está contenido dentro de un paquete y se desea importar, se hace uso de la declaración `from`, la cual indica la ruta hacia el módulo dentro del paquete.

Un ejemplo sobre como importar un módulo podría ser el siguiente:

```
1 from datetime import date
```

Código 3.1. Ejemplo de importación de un módulo

En este ejemplo, se importa el módulo `date` del paquete `datetime`.

También es posible renombrar el módulo cuando se importa, esto se realiza utilizando la declaración `as`. Por ejemplo:

```
import datetime as dt
```

Código 3.2. Ejemplo de importación de un módulo con renombramiento.

En este ejemplo, se importa el módulo (que es en si mismo un paquete con diversos submódulos) `datetime` y se le asocia el nombre `dt`.

Cuando un paquete cuenta con subpaquetes, es posible importar un módulo dentro de estos utilizando un punto como notación para navegar dentro de los subdirectorios. Para realizar esto es importante que cada subdirectorio cuente con un archivo `__init__.py` en el cual se importen los módulos que contiene. Esto convierte al directorio en un módulo en sí mismo, por lo que podemos decir en que realidad se están importando submódulos de un paquete. Por ejemplo:

```
import scipy.datasets as dtst
from scipy.signal import find_peaks
```

Código 3.3. Ejemplo de importación de submódulos.

En este ejemplo, se está importando el submódulo `datasets` de la biblioteca `scipy` con el nombre `dtst`, y posteriormente se importa la función `find_peaks` contenida en el submódulo `signal` de la biblioteca `scipy`.

Para poder importar un módulo, este debe estar contenido dentro del mismo directorio que el archivo de código al que se desea importar. Si este módulo pertenece a un paquete, el directorio completo del paquete debe estar contenido dentro del mismo directorio que el archivo de código. Para evitar esto, los módulos y paquetes pueden ser instalados como librerías, como se verá en la secciones 3.4 y 3.5.

3.2.6. Funciones

Una *función* es un bloque de código asociado a una declaración de función. Una función toma una cierta cantidad de argumentos (objetos) y ejecuta el código utilizando los mismos. Finalmente retorna un resultado asociado a los argumentos y la lógica con la que fueron operados. También es posible que la función no tenga un retorno, y que su objetivo sea únicamente el procedimiento llevado a cabo en el código.

Para declarar una función se utiliza la declaración `def`. Un ejemplo de una declaración de una función es el siguiente:

```
def div(num,den):
    r = num/den
    return r
```

```
4
5  a = div(10,2)
6  print(a)
7  # out: 5
```

Código 3.4. Ejemplo de declaración de una función.

En este ejemplo se declara una función cuyos argumentos representan el numerador y denominador de una división, la función ejecuta una división de estos argumentos y retorna el resutlado. Se ejemplifica su uso con los argumentos `10` y `2`, retornando así el número `5`. Notar que el orden en el que los argumentos se colocan es importante, aunque también es posible asociar palabras clave a cada argumento de modo que se pueda prescindir del orden.

3.2.7. Clases

En Python es posible definir un objeto distinto a los que están programados dentro del intérprete. A estos objetos se les llama *clases*, a las cuales es posible definirles atributos y métodos. Para crear una clase se debe definir su estructura y comportamiento, y posteriormente es posible crear objetos de esta clase, a estos se les llamará instancias.

Un atributo no es más que un espacio de memoria asociado a una instancia de la clase. Este, al igual que otro objeto, almacena datos, y es posible definir el tipo de objeto que será almacenado en este atributo. Una clase puede contener múltiples atributos, y estos pueden ser de distinto tipo de objeto. Los atributos pueden ser específicos para cada instancia o pueden ser generales para toda la clase, dependiendo de esto se les conoce como atributos de instancia o de clase.

Por ejemplo, un atributo de instancia podría ser el nombre asociado a esa instancia, pero un atributo de clase podría ser un conteo de la cantidad de instancias que existen de la misma.

También es posible definir funciones asociadas a una clase, a estas se les conoce como métodos, y dependiendo de si los métodos son dependientes de la instancia en particular o no, se les conoce como métodos de instancia o métodos de clase respectivamente.

Por ejemplo, un método de instancia podría imprimir el nombre de dicha instancia en la terminal y retornar un `str` del mismo. Mientras que un método de clase podría imprimir la cantidad de instancias que existen y retornar un `int` de este dato.

3.3. ¿Qué es una biblioteca?

En computación, una *biblioteca* hace referencia a una colección de recursos utilizados por un programa de computadora. En los 40s, las tiendas de programación tenían bibliotecas reales de código que contenían carretes de cinta, los programadores visitaban estas bibliotecas para cargar los códigos en sus programas [22].

3.4. ¿Qué es una biblioteca de Python?

Una biblioteca de Python es un conjunto de paquetes y módulos de Python que pueden ser reutilizados. Estas son instaladas como parte del intérprete lo que permite acceder a sus funciones a desde cualquier parte del código.

Se accede a los contenidos de una biblioteca importándola en el código utilizando la sintaxis para importar módulos: `import library_name`, donde `library_name` es el nombre de la librería que se desea utilizar.

3.5. Estructura y configuración de una biblioteca de Python

Una biblioteca de Python puede ser generada a partir de cualquier paquete o módulo creado por un usuario. Para esto es necesario realizar algunas configuraciones además de seguir una estructura para mantener el orden.

3.5.1. Estructura de una biblioteca de Python

Las bibliotecas están estructuradas por medio de paquetes, subpaquetes, módulos y submódulos. Dependiendo de la complejidad y la extensión de la biblioteca esta puede recurrir a todos estos o no.

Esencialmente una biblioteca debe contener al menos un módulo, pero no se limita a esto. Se recurre a separar en submódulos o subpaquetes cuando estos son de distinta naturaleza. Por ejemplo, en la librería `pydyn_surv` que se detalla en la sección 3.6, se tiene dos componentes con objetivos muy diferentes entre sí. El paquete `classes` contiene la estructura operativa del modelo, mientras que el paquete `LinearClassifier` contiene las funciones de machine learningque se requieren para entrenar el modelo.

También es importante conocer algunos nombres clave de archivos u objetos, ya que el intérprete reconocerá a los mismos y son útilies para agregar funcionalidad al paquete [35].

3.5.1.1. ___name___

Este es un atributo que almacena un `str` con el nombre del módulo al que está asociado.

3.5.1.2. ___main___ y ___main___.py

Cuando un programa de Python es ejecutado, al módulo que se ejecuta se le asocia el nombre `__main__`, esto es útil, ya que otros módulos que sean importados por el programa tendrán un nombre distinto, esto distingue al módulo que está siendo ejecutado de los que están siendo importados, y permite colocar una función dentro de los módulos cuyos contenidos se ejecuten únicamente cuando el módulo en sí esté siendo ejecutado y no cuando está siendo importado por otro módulo.

El archivo `__main__.py` se coloca dentro del directorio de un paquete, cuando este paquete es ejecutado como un programa, el intérprete ejecutará el código contenido el mismo. Sin embargo, si el paquete es importado desde otro módulo este código no será ejecutado.

3.5.1.3. ___init___.py

El archivo `__init__.py` tiene una funcionalidad parecida al archivo `__main__.py`, pero esta vez el código es ejecutado cuando el paquete es importado y no ejecutado. El archivo se coloca dentro del directorio de un paquete, cuando el paquete es importado por un módulo, el intérprete ejecutará el código contenido en el mismo.

3.5.1.4. ___pycache___

Este es un directorio que se crea cuando un módulo es ejecutado. Dentro de él se almacena la versión "compilada" del mismo con el nombre de `módulo.versión.pyc`. Esto se hace con el fin de minimizar el tiempo que lleva cargar dicho módulo. Es importante notar que estos archivos solamente son una abstracción del código optimizado para el intérprete, más no son binarios compilados per se.

3.5.2. Configuración de una biblioteca de Python

Para que un paquete pueda volverse una biblioteca, este debe configurarse para poder ser instalado en el intérprete. La instalación puede realizarse utilizando un administrador de paquetes para Python como `pip`.

La configuración se maneja en un archivo contenido dentro del directorio del paquete con el nombre `setup.py`.

3.5.2.1. setup.py

Dentro de este archivo se deben colocar las configuraciones pertinentes a la biblioteca. Originalmente se utilizaba `Distutils` para realizar la configuración de la biblioteca. Ahora se utiliza `setuptools`, el cual tiene una funcionalidad más amplia aunque este no es parte de la biblioteca estándar de Python [16] [33].

El soporte para el uso del script `setup.py` de forma directa (es decir, ejecutar `python setup.py`) ha sido removido, y se espera que se realice una transición a otros formatos de configuración como lo son `pyproject.toml` y `setup.cfg`. Sin embargo, la funcionalidad de estos formatos es más limitada y aún está pobremente documentada. A pesar de esto, el uso del script `setup.py` aún es permitido en administradores de paquetes como `pip` (ver sección 3.5.3.3) [10].

Un ejemplo sencillo de un archivo de configuración, expuesto en la documentación oficial de `setuptools` [34] es el siguiente:

```
from setuptools import setup

setup(
    name='mypackage',
    version='0.0.1',
    install_requires=[
        'requests',
        'importlib-metadata; python_version == "3.8"',
    ],
)
```

Código 3.5. Ejemplo de archivo de configuración de una biblioteca de Python. Fuente: Python Packaging Authority [34]

En este archivo de configuración se define el nombre de la biblioteca como `'mypackage'`, la versión en la que se encuentra (existe una forma estándar para numerar versiones, se detallará en la sección 3.5.2.2), y las bibliotecas que requiere el programa para funcionar.

Además de estas definiciones también es posible agregar otras como lo son: una descripción de la biblioteca, un enlace hacia la biblioteca, el autor, la licencia, los paquetes contenidos dentro de la biblioteca, entre otros.

3.5.2.2. Numeración de versiones

Para llevar un registro sobre los cambios implementados en una biblioteca, es buena práctica seguir algún criterio para numeración de la versión de la biblioteca.

El criterio más comúnmente utilizado es el siguiente: se realiza una numeración que cuenta con tres números, separados por puntos, cada uno de estos números representa un tipo de cambio de versión en particular [32]:

```
[Mayor].[Menor].[Arreglo de fallo]
```

El primer número representa en valor de la versión mayor. Cuando se lanza una nueva versión de una biblioteca y esta contiene cambios drásticos, mejoras considerables, cambios en la funcionalidad o sintaxis, esto se conoce como un lanzamiento mayor y representaría un aumento de una unidad para el número de versión mayor. Cuando se realiza un cambio de versión mayor los demás números son restablecidos a cero.

El segundo número representa el valor de la versión menor. Cuando se lanza una nueva versión de una biblioteca y esta contiene cambios pequeños, retrocompatibles (que no cambian la sintaxis o la usabilidad de la biblioteca), mejoras menores y entre otros cambios pequeños, se denomina lanzamiento menor. Este implicaría un aumento de una unidad para el segundo número de versión. Cuando se realiza un lanzamiento de versión menor, el conteo del último número (arreglo de fallo) se debe restablecer a cero.

Finalmente, cuando se realizan arreglos de fallos en el código y se corrigen errores mínimos que no se percibieron previamente, entonces se realiza un lanzamiento de arreglo de fallo. Esto implica un aumento de una unidad en el tercer número de versión.

Por ejemplo, la versión más reciente de `numpy` es la siguiente: `1.24.2`. Esto queire decir que cuando se han realizado dos lanzamientos mayores (ya que se empieza a contar desde cero) con cambios drásticos, veinticinco lanzamientos menores a partir de la última versión mayor (versión 1) y dos arreglos de fallos a partir de la última versión menor (versión 24).

3.5.3. Otras herramientas

Para realizar una biblioteca también se hace uso de otras herramientas, estas son útiles para mantener la organización adecuada, llevar un registro del código y sus cambios, mantener una documentación actualizada de las funcionalidades de la biblioteca e instalar y probar la biblioteca y los cambios y mejoras que sufra.

3.5.3.1. Git

Git es un sistema de control de versiones, este es gratuito y de código abierto y es mundialmente usado por desarrolladores para el manejo y control de versiones [6].

Git almacena un historial de los cambios realizados en el código, permite realizar variaciones, restablecer el código desde un punto funcional. Mantiene el orden para el almacenamiento e historial del código, además de ser liviano y rápido.

3.5.3.2. Pdoc

pdoc es un programa sencillo que genera documentación de una biblioteca, siguiendo la estructura de la misma. Dentro del código, se pueden colocar textos de documentación conocidos como *docstrings* (a menudo estos suelen ser confundidos con comentarios, ya que el intérprete los ignora). Para realizar un docstring se escribe el texto de documentación deseado dentro de una triple comilla doble como se muestra en el ejemplo 3.6.

```
1 count:int = 0
2 """
3 Contador inicializado en cero.
4 """
```

Código 3.6. Ejemplo de docstring en Python.

En este ejemplo vemos que se define el objeto tipo `int` con el nombre `count` igualado a cero. En este caso el docstrig es la descripción del objeto `Contador inicializado en cero`.

Cuando un módulo o paquete contiene docstrings, estos serán automáticamente ubicados por `pdoc` para ser agregados a la documentación generada. La documentación se genera en un archvo `.html` que puede ser desplegado en una página web. La forma de generar la documentación usando `pdoc` en una terminal se desarrolla en el ejemplo 3.7.

```
pdoc path/module.py -o ./doc
```

Código 3.7. Generación de documentación con `pdoc`.

Donde `path/module.py` es la ruta al módulo del que se desea generar la documentación, aquí también puede colocarse la ruta a un paquete si se desea generar la documentación de todo el paquete. Por otro lado `path/doc` es la ruta del directorio de salida donde se guardará el archivo `.html` generado.

Es posible generar documentación más compleja utilizando una sintaxis para esta que pueda ser entendida por `pdoc`. Los formatos aceptados son `Numpy` y `Google`, además se puede utilizar código de `Markdown` y este será compilado.

3.5.3.3. Pip

pip es un administrador de paquetes escrito en Python y diseñado para instalar y administrar paquetes de software escritos en Python [19].

Cuando se instalan un paquete con pip, este se encargará de chequear si el mismo ya se encuentra instalado al igual que sus dependencias, instalará las dependencias que hagan falta y posteriormente el paquete si no estaba instalado.

3.5.3.4. GitHub

GitHub es un servicio de alojamiento de código en la nube basado en git (ver sección 3.5.3.1). Esta plataforma permite compartir código y trabajar de forma colaborativa. Además de ofrecer planes corportativos de pago, GitHub es utilizado principalmente por desarrolladores en su versión gratuita [21]. Esto hace de GitHub uno de los sitios más utilizados para compartir y publicar código libre.

3.6. Desarrollo de la biblioteca pydyn-surv

Como parte de la investigación, se desarrolló una biblioteca en Python que engloba el modelo desarrollado en la sección 2.2. A esta se le denominó *pydyn-surv*, obteniendo su nombre de las siglas: “py”, “dyn” y “surv” en representación del lenguaje utilizado (Python) y el término dynamic survey (cuestionario dinámico) respectivamente.

La biblioteca pydyn-surv fue desarrollada en un sistema operativo Linux, haciendo uso de un ambiente virtual para llevar un registro de las dependencias. Se

utilizó git como gestor de versiones y github como servicio de alojamiento en la nube. La biblioteca se encuentra disponible de forma pública en esta plataforma.

3.6.1. Estructura de pydyn-surv

Se desarrolló esta biblioteca en submódulos con la siguiente estructura:

```
pydyn_surv
├── __init__.py
├── funcs.py
├── item.py
├── ml.py
├── other_classes.py
└── survey.py
```

El archivo `__init__.py` importa los cinco submódulos, haciéndolos accesibles desde la biblioteca. Los submódulos contienen las clases y funciones necesarias para construir el cuestionario dinámico y entrenar el modelo, estos se describirán a continuación.

3.6.1.1. Submódulo item

Este submódulo contiene la definición de la clase `item`, además de un diccionario con los parámetros por defecto de esta clase. La clase `item` abstrae la definición de un ítem del cuestionario. Se encarga de almacenar la información referente al mismo. Dentro de sus atributos se encuentra el enunciado del ítem, una lista con las posibles respuestas, además de los valores asociados a estas respuestas utilizados posteriormente en el entrenamiento, el vector de características del ítem, el cuestionario al que el ítem pertenece, el conteo de veces que ha sido respondido, el historial de respuestas, la función de probabilidad asociada al mismo, la etiqueta predicha, entre otros.

Dentro de la clase `item` se define el método `answer` que se encarga de almacenar la respuesta del ítem, además de actualizar el conteo e historial de respuestas. En caso de tener un cuestionario asociado, actualiza el conteo e historial de respuestas de la categoría o categorías a las que pertenece, y ejecuta el proceso de entrenamiento para ese cuestionario con el training data set actualizado.

La clase `item` se inicializa por defecto utilizando el diccionario `DEFAULT_PARAMETERS_DICT` que se incluye en este módulo, aunque es posible cambiar los parámetros de inicialización. Dentro de este diccionario debe definirse una pregunta con la clave

'question', y una lista de respuestas con la clave 'answers', estas están vacías por defecto. También se debe asignar una etiqueta a cada respuesta como se hizo en la tabla 2.1, esto se realiza con la clave 'answers_values'. En la sección 3.7.2.1 se detalla sobre la sintaxis y el uso de esta clase, la misma incluye un ejemplo de uso.

3.6.1.2. Submódulo survey

Este submódulo contiene la definición de la clase survey, la cual abstrae la definición de cuestionario en este modelo. Se encarga de almacenar la información referente al cuestionario, además de los ítems que lo componen. Dentro de sus atributos se encuentra el nombre del cuestionario, su origen y descendencia, una lista de ítems que lo componen, las categorías que lo componen, la categoría a la que pertenece en su origen, el conteo de veces que algún ítem que pertenezca a este ha sido respondido, los historiales de respuesta y conteos por categoría, la función de probabilidad asociada al mismo, la etiqueta predicha, el peso asociado al cuestionario, entre otros.

Dentro de la clase survey se define el método train que se encarga de entrenar el peso asociado. Antes de realizar el proceso de entrenamiento realiza una actualización que verifica que el training data set contenga todos los elementos.

La clase survey se inicializa por defecto sin ítems asociados, con un nombre vacío, sin un training data set previo asociado, sin un peso inicializado (en este caso el peso se inicializa en un vector con la dimensión del cuestionario con ceros en cada coordenada), con la función de probabilidad, condición y entrenamiento por defecto, y con las listas de categorías, origen y descendencia vacías. En la sección 3.7.2.2 se detalla sobre la sintaxis y el uso de esta clase. La misma incluye un ejemplo de uso.

3.6.1.3. Submódulo ml

Este submódulo contiene las funciones relacionadas con el proceso de aprendizaje y los algoritmos de machine learning. Entre estas funciones se encuentran los predictores de clasificación lineal y regresión, las funciones de pérdida y las funciones de descenso del gradiente. En la sección 3.7.2.3 se detalla sobre la sintaxis y el uso de estas funciones.

3.6.1.4. Submódulo funcs

Este submódulo contiene herramientas útiles para el modelo, entre estas se encuentran las funciones de cálculo de probabilidad y condición, además de la función

entrenamiento por defecto que hace uso del submódulo ml expuesto en la sección anterior. En la sección 3.7.2.4 se detalla sobre la sintaxis y el uso de estas funciones.

3.6.1.5. Submódulo other_classes

Este submódulo se creó con el propósito de definir otras clases útiles para el uso del modelo distintas a las clases `item` y `survey`. Este contiene la clase `pydyn_surv_list` que es una clase que heredada de la clase `list` de Python con algunos métodos útiles añadidos. Esta clase está diseñada para almacenar una lista de objetos de la clase `item` o `survey`. En la sección 3.7.2.5 se detalla sobre la sintaxis y el uso de esta clase.

3.7. Documentación de uso

3.7.1. Instalación

Para instalar la biblioteca `pydyn_surv` es necesario tener instalado Python y `pip` en el sistema. Además de tener instalado `git` para poder clonar el repositorio de la biblioteca.

El proceso de instalación consiste en clonar el repositorio de la biblioteca, y luego instalarla utilizando `pip`, esto puede realizarse en una terminal con los comandos expuestos en el ejemplo 3.8.

```
1 git clone https://github.com/JARA99/Dynamic_Survey_lib.git
2 pip install ./Dynamic_Survey_lib
```

Código 3.8. Instalación de `pydyn_surv`.

3.7.2. Uso de la biblioteca pydyn-surv

En esta sección se presenta la documentación de uso de los métodos y funciones principales de la biblioteca `pydyn_surv`, además de incluir algunos ejemplos de aplicación. Se cuenta con una documentación mucho más extensa que cubre todos los métodos y funciones de la biblioteca, la cual está disponible de forma pública en el repositorio de la biblioteca. Esta fue realizada utilizando la herramienta `pdoc` expuesta en la sección 3.5.3.2.

La documentación completa puede accederse a través del enlace `https://jara99.github.io/Dynamic_Survey_lib/`.

3.7.2.1. Clase item

Para importar la clase `item` se realiza lo siguiente:

```
1 from pydyn_surv.item import item
```

Código 3.9. Importación de la clase `item`.

El constructor de esta clase está dado por:

`pydyn_surv.item.item()`

Crea una instancia de un ítem.

Parámetros:

- **parameters_dict** (`dict`): Diccionario que contiene los atributos para esta instancia, por defecto `pydyn_surv.item.DEFAULT_PARAMETERS_DICT`.
- **id_** (`int`): Id para el ítem, por defecto `None`.
- **origin_survey** (`pydyn_surv.survey`): Cuestionario de origen para el ítem, por defecto `None`.
- **probability_function** (`callable`): Función de probabilidad para el ítem, por defecto `pydyn_surv.funcs.FUNC_LIKERT_ITEM_PROBABILITY`.

Retorna:

- (`pydyn_surv.item.item`): Una instancia de ítem con los parámetros de inicialización.

La mayoría de métodos se utilizan para asignar, actualizar u obtener atributos del ítem. Uno de los métodos más importantes de la clase es `answer`, este añade funcionalidad a la clase permitiendo registrar las respuestas de los ítems cuando son contestados, está dado por:

`pydyn_surv.item.item.answer()`

Si la respuesta está en el rango aceptado, se registra en el historial y se aumenta en uno el contador de lanzamientos. Si la respuesta está fuera de rango, no se registra a menos que el parámetro `force=True`.

Parámetros:

- **answer** (`float`): La respuesta dada por el usuario.
- **force** (`bool`): Si es `True` se registra la respuesta aunque esté fuera de rango, por defecto `False`.

En el ejemplo 3.10 se detalla el uso de esta clase, comenzando con la definición de algunos atributos de clase, pasando por la creación de una instancia y finalizando con la ejecución de algunos métodos.

```
1  from pydyn_surv.item import item
2
3  # Definimos un diccionario con la información del ítem
4  item_dict = {
5      'question':'El invierno me gusta más que el verano',
6      'answers':['Muy en desacuerdo','En desacuerdo','Neutral','De
       acuerdo','Muy de acuerdo'],
7      'answers_values':[-2,-1,0,1,2],
8      'category_vector':[1,-1],
9      'expert_extra':0,
10     'id':1
11 }
12
13 # Definimos una instancia de item utilizando los datos del diccionario
14 item_instance = item(item_dict)
15
16 # Contestamos al item con un punteo de -1 que corresponde a 'En
       desacuerdo'
17 item_instance.answer(-1)
18
19 # Imprimimos información del item
20 item_instance.print_info()
21
22 # Obtenemos el set de entrenamiento y lo imprimimos
23 print('\nEl set de entrenamiento
       es:\n{}'.format(item_instance.get_dataset_history()))
```

Código 3.10. Ejemplo de uso de la clase `item`

3.7.2.2. Clase survey

Para importar la clase `survey` se realiza lo siguiente:

```
1 from pydyn_surv.survey import survey
```

Código 3.11. Importación de la clase `survey`.

El constructor de esta clase está dado por:

`pydyn_surv.survey.survey()`

Crea una instancia de un cuestionario.

Parámetros:

- **items** (`list`): La lista de ítems que el cuestionario contendrá, por defecto `[]`. Los ítems pueden ser agregados luego con los métodos `set_items` y `add_item`. Los ítems pueden ser instancias de `pydyn_surv.item.item` o diccionarios con los parámetros para crearlos.
- **name** (`str`): El nombre del cuestionario, por defecto `''`. También se usa para establecer el nombre de la categoría de origen en caso de que no se establezca con el parámetro `origin_category`.
- **init_training_dataset** (`list`): El conjunto de entrenamiento inicial, por defecto `None`.
- **w** (`np.ndarray`): El vector de pesos inicial, por defecto `None`. Si no se indica, se establecerá como un vector de ceros con la misma dimensión que los ítems.
- **predictor** (`callable`): La función de predicción, por defecto `pydyn_surv.funcs.PREDICTOR`. Puede ser cambiada después con el método `set_predictor`.
- **launch_format** (`list`): El formato para mostrar los ítems en la terminal, por defecto `pydyn_surv.survey.LAUNCH_FORMAT`.
- **categories** (`list`): La lista de categorías que el cuestionario contendrá, por defecto `[]`. Las categorías pueden ser agregadas luego con los métodos `set_categories` y `add_category`.
- **origin** (`list`): La lista de cuestionarios que son el origen del cuestionario, por defecto `[]`. El origen puede ser cambiado luego con el método `set_origin`.

- **offspring** (`list`): La lista de cuestionarios que pertenecen a la descendencia del cuestionario, por defecto `[]`. Estos pueden ser cambiados con el método `set_offspring`.
- **origin_category** (`str`): El nombre de la categoría en el origen, por defecto `None`. Si no se indica, se establecerá como el nombre del cuestionario.
- **condition_function** (`callable`): La función de condición, por defecto `pydyn_surv.funcs.FUNC_TRUE`. Puede ser cambiada con el método `set_condition_function`.
- **probability_function** (`callable`): La función de probabilidad, por defecto `pydyn_surv.funcs.FUNC_TRUE`. Puede ser cambiada con el método `set_probability_function`.
- **train_function** (`callable`): La función de entrenamiento, por defecto `pydyn_surv.funcs.TRAIN_FUNCTION`. Puede ser cambiada con el método `set_train_function`.

Retorna:

- (`pydyn_surv.survey.survey`): La instancia del cuestionario.

Aunque la mayoría de métodos de esta clase son utilizados para asignar, actualizar u obtener atributos de la instancia, existen otros que añaden funcionalidad, como el método de entrenamiento `train` que se ejecuta automáticamente cada vez que un ítem del cuestionario es contestado:

`pydyn_surv.survey.survey.train()`

Entrena el cuestionario utilizando la función de entrenamiento asignada, por defecto utiliza `pydyn_surv.funcs.TRAIN_FUNCTION`, pero puede ser cambiada con el método `set_train_function`.

Parámetros:

- **args** (`list`): Los argumentos para la función de entrenamiento.
- **kwargs** (`dict`): Los argumentos con nombre para la función de entrenamiento.

Otro método importante es el que se utiliza para obtener un ítem del cuestio-

nario de forma aleatoria utilizando las probabilidades de cada ítem `launch_random`:

`pydyn_surv.survey.survey.launch_random()`

Retorna un ítem del cuestionario de forma aleatoria utilizando las probabilidades de cada ítem. El ítem es seleccionado utilizando la función `random.choices` por defecto, pero puede ser cambiada con el parámetro `random_func`.

Parámetros:

- **random_func** (`callable`, opcional): La función a utilizar para seleccionar un ítem aleatorio, por defecto `random.choices`.
- **all_zero_to_one** (`bool`, opcional): Si es `True` y las probabilidades de todos los ítems son iguales a cero, estas se establecen igual a uno. Por defecto `False`.

Retorna:

- **item_** (`pydyn_surv.item.item`): El ítem seleccionado aleatoriamente.
- **item_.question_text** (`str`): El texto de la pregunta del ítem seleccionado aleatoriamente.
- **item_.answers_text** (`list`): El texto de las respuestas del ítem seleccionado aleatoriamente.
- **item_.answers_values** (`list`): El valor de las respuestas del ítem seleccionado aleatoriamente.

Finalmente un método que podría ser de utilidad en especial para hacer pruebas del cuestionario es `launch_on_terminal`, que permite desplegar un ítem en la terminal y contestarlo:

`pydyn_surv.survey.survey.launch_on_terminal()`

Despliega un ítem en la terminal, incluyendo su enunciado y posibles respuestas, espera el input del usuario para responder dicho item. Si el parámetro `force_answer` es `True`, fuerza la respuesta sin verificar el input.

Parámetros:

- **item__** (`pydyn_surv.item.item`): El ítem a desplegar.
- **force__answer** (`bool`, opcional): Si es `True`, fuerza la respuesta sin verificar el input, por defecto `False`.

En el ejemplo 3.12 se detalla el uso de esta clase, comenzando por la definición de algunos parámetros de los ítems contenidos en el cuestionario y posteriormente haciendo uso de los métodos de clase más comunes:

```
from pydyn_surv.survey import survey

# Para definir un cuestionario primero definimos una lista de
    diccionarios con la información de los ítems en este.
# Se utilizan las siguientes categorias

categories = ['Invierno','Primavera','Verano','Otoño']

# Definimos un diccionario con la información del primer ítem
i1_dict = {
    'question':'El invierno me gusta más que el verano',
    'answers':['Muy en desacuerdo','En desacuerdo','Neutral','De
    acuerdo','Muy de acuerdo'],
    'answers_values':[-2,-1,0,1,2],
    'category_vector':[1,0,-1,0],
    'expert_extra':0,
    'id':0
}

# Definimos un diccionario con la información del segundo ítem
i2_dict = {
    'question':'Me gusta la primavera más que el otoño o el verano',
    'answers':['Muy en desacuerdo','En desacuerdo','Neutral','De
    acuerdo','Muy de acuerdo'],
    'answers_values':[-2,-1,0,1,2],
    'category_vector':[0,1,-1,-1],
    'expert_extra':0,
```

```
    'id':1
}

# Generamos la lista de diccionarios con la información de los ítems
items_list = [i1_dict,i2_dict]

# Definimos una instancia de survey utilizando la lista de diccionarios
survey_instance = survey(items_list,name='Ejemplo de
    cuestionario',categories=categories)

# Imprimimos información del survey
survey_instance.print_info()

# Obtenemos el primer item utilizando el diccionario de items del
    cuestionario
i1 = survey_instance.item_by_id[0]

# Obtenemos el segundo item utilizando la posición en la lista de
    items del cuestionario
i2 = survey_instance.get_items()[1]

# Contestamos al primer item con un punteo de -1 que corresponde a 'En
    desacuerdo'
i1.answer(-1)

# Contestamos al segundo item con un punteo de 2 que corresponde a
    'Muy de acuerdo'
i2.answer(2)


# Obtenemos el set de entrenamiento del cuestionario y lo imprimimos
print('\nEl set de entrenamiento
    es:\n{}'.format(survey_instance.get_training_dataset()))

# Obtenemos el historial de respuestas por categoría y lo imporimimos
print('\nEl historial de respuestas por categoría
    es:\n{}'.format(survey_instance.get_category_answer_history()))

# Contestamos una vez más los items
i1.answer(-2)
i2.answer(1)

# Imprimimos información del survey, ahora el training dataset
```

```
    mostrará el hisotrial de respuestas con vectores de categorías y
    se tendrán etiquetas predichas y calculadas
survey_instance.print_info()

# También es posible obtener un item de dorma pseudoaleatoria en donde
    se tome en cuenta la probabilidad calculada de cada item
ir,ir_text,ir_answerstext,ir_answersvals =
    survey_instance.launch_random()

# Imprimimos el item obtenido
print('\nSe obtuvo el ítem: {},\nCuyo enunciado es: {},\nSus posibles
    respuestas son: {},\nY los valores asociados son:
    {}'.format(ir.id,ir_text,ir_answerstext,ir_answersvals))

# Este item puede ser contestado, se podría usar el método answer()
    utilizado antes, pero se ejemplifica ahora el uso del método
    launch_on_terminal()
survey_instance.launch_on_terminal(ir)

# Volvemos a imprimir información del survey
survey_instance.print_info()

# Este fue un ejemplo básico del uso de la clas survey
```

Código 3.12. Ejemplo de uso de la clase `survey`

3.7.2.3. Submódulo ml

Para hacer uso de la biblioteca no es necesario importar este submódulo, sin embargo este podría ser útil en caso de querer cambiar las definiciones por defecto de las funciones de pérdida y descenso del gradiente.

La función de pérdida utilizada por defecto es la función squared-loss definida en la ecuación 1.11, esta función se define en la librería de la siguiente manera:

`pydyn_surv.ml.squared_loss()`

Retorna la función de pérdida squared-loss dados un vector de peso, un vector de características y una etiqueta asociada.

Parámetros:

- **w** (`np.ndarray`): Vector de pesos de dimensión n.
- **x** (`np.ndarray`): Vector de características de dimensión n.
- **y** (`float`): Etiqueta asociada a **x**.

Retorna:

- **float**: Valor de la función de pérdida squared-loss.

Por otro lado, el algoritmo de descenso del gradiente utilizado es el descenso del gradiente por bloque expuesto en la sección 1.4.8.1. Este algoritmo se define en la librería de la siguiente manera:

`pydyn_surv.ml.gradient_descent()`

Ejecuta el algoritmo de descenso del gradiente por bloque para todo el training data set y retorna un vector de pesos entrenado.

Parámetros:

- **loss** (`function`): Función de pérdida.
- **dd_loss** (`function`): Función derivada de la función de pérdida.
- **training_dataset** (`Iterable`): Una lista que contiene la pareja (peso,etiqueta) para el entrenamiento.
- **eta** (`float`, opcional): El tamaño del paso, por defecto 0.01.
- **iterations** (`int`, opcional): El número de iteraciones, por defecto 500.
- **verbose** (`bool`, opcional): `True` para verboso, por defecto `True`.
- **w** (`np.ndarray`, opcional): Un vector de pesos inicial, por defecto `None`.

Retorna:

- (`np.ndarray`): Vector de pesos entrenado.

3.7.2.4. Submódulo funcs

Este submódulo contiene diversas funciones útiles para fabricar el cuestionario. A continuación se presenta la documentación de algunas de estas:

Las funciones `FUNC_TRUE` y `FUNC_FALSE` se definen de la siguiente manera:

pydyn_surv.funcs.FUNC_FALSE()

Retorna `False`. Útil para los métodos `condition` y `probability` de los objetos `pydyn_surv.survey.survey` y `pydyn_surv.item.item`.

Retorna:

- (`bool`): `False`.

pydyn_surv.funcs.FUNC_TRUE()

Retorna `True`. Útil para los métodos `condition` y `probability` de los objetos `pydyn_surv.survey.survey` y `pydyn_surv.item.item`.

Retorna:

- (`bool`): `True`.

La función de probabilidad por defecto utiliza la definición planteada en la ecuación 2.2. Esta función está incorporada en la librería de la siguiente manera:

pydyn_surv.funcs.FUNC_LIKERT_ITEM_PROBABILITY()

Retorna la probabilidad de que un ítem sea lanzado basado en la etiqueta predicha y la traslación del eje solamente.

Parámetros:

- **axis_move** (`int`): El movimiento del eje.

Retorna:

- (`float`): La probabilidad de que el ítem sea lanzado.

También se incluye la función `FUNC_LIKERT_SURVEY_PROBABILITY_WITH_STATISTICS`, que está basada en la ecuación 2.3.

pydyn_surv.funcs.FUNC_LIKERT_SURVEY_PROBABILITY_WITH_STATISTICS()

Retorna la probabilidad de que un ítem sea lanzado basado en la etiqueta predicha, la traslación del eje y las estadísticas del historial del ítem y de la categoría.

Parámetros:

- **axis_move** (int): El movimiento del eje.
- **not_repeated_since** (int): El número mínimo de lanzamientos desde el último lanzamiento del ítem.
- **std_weight** (float): El peso de la desviación estándar del historial del ítem en la probabilidad.
- **cat_std_weight** (float): El peso de la desviación estándar del historial de la categoría en la probabilidad.
- **launch_count_weight** (float): El peso del conteo de lanzamientos del ítem en la probabilidad.
- **cat_launch_count_weight** (float): El peso del conteo de lanzamientos de la categoría en la probabilidad.
- **predicted_label_weight** (float): El peso de la etiqueta predicha en la probabilidad.

Retorna:

- (float): La probabilidad de que el ítem sea lanzado.

3.7.2.5. Clase pydyn_surv_list

Esta clase se hereda de una lista de Python, agregando algunos métodos útiles para acceder a la información de los objetos `item` y `survey` que contiene. Para importarla se utiliza el siguiente comando:

```
1 from pydyn_surv.other_classes import pydyn_surv_list
```

Código 3.13. Importación de la clase `pydyn_surv.other_classes.pydyn_surv_list`

El constructor de esta clase está definido de la siguiente manera:

```
1 class pydyn_surv_list(list):
```

```
2       def __init__(self,l:list = []):
3           super().__init__(l)
```

Código 3.14. Constructor de la clase `pydyn_surv.other_classes.pydyn_surv_list`

A simple vista esta clase es una copia de la clase `list` de Python, pero esta cuenta con los siguientes métodos adicionales:

`pydyn_surv.other_classes.pydyn_surv_list.probabilities()`

Retorna una lista de probabilidades de cada ítem en la lista.

Parámetros:

- **all_zero_to_one** (`bool`): Si es `True`, todas las probabilidades son establecidas a 1 si todas son 0.
- **nan_to_zero** (`bool`): Si es `True`, todas las probabilidades que son `np.nan` son establecidas a 0.
- ***args, ****kargs**: Argumentos y argumentos de palabras clave para ser pasados al método `probability` de cada ítem.

Retorna:

- (`list`): Una lista de probabilidades de cada ítem en la lista.

`pydyn_surv.other_classes.pydyn_surv_list.ids()`

Retorna una lista de ids de cada ítem en la lista (solo funciona si la lista está hecha de objetos `pydyn_surv.item.item`).

Retorna:

- (`list`): Una lista de ids de cada ítem en la lista.

`pydyn_surv.other_classes.pydyn_surv_list.names()`

Retorna una lista de nombres de cada ítem en la lista (solo funciona si la lista está hecha de objetos `pydyn_surv.survey.survey`).

Retorna:

- (`list`): Una lista de nombres de cada ítem en la lista.

pydyn_surv.other_classes.pydyn_surv_list.questions()

Retorna una lista de preguntas de cada ítem en la lista (solo funciona si la lista está hecha de objetos pydyn_surv.item.item).

Retorna:

- (list): Una lista de preguntas de cada ítem en la lista.

pydyn_surv.other_classes.pydyn_surv_list.answer_history()

Retorna una lista de answer_history de cada ítem en la lista (solo funciona si la lista está hecha de objetos pydyn_surv.item.item).

Retorna:

- (list): Una lista de answer_history de cada ítem en la lista.

4. Ejemplo de aplicación

Como prueba de concepto, se ha realizado un ejemplo de aplicación de la metodología propuesta, en este capítulo se desarrolla dicho ejemplo.

4.1. Descripción del ejemplo

Se propone desarrollar un cuestionario dinámico sobre pasatiempos o "hobbies" debido a la diversidad que engloba este tema. Se hace uso de la metodología de cuestionarios múltiples expuesta en la sección 2.2.5, por lo que en este capítulo se denominará "cuestionario" a cada uno de los cuestionarios desarrollados utilizando esta metodología; y al ejemplo en sí, que engloba todos los cuestionarios, se le denominará "Cuestionario Dinámico".

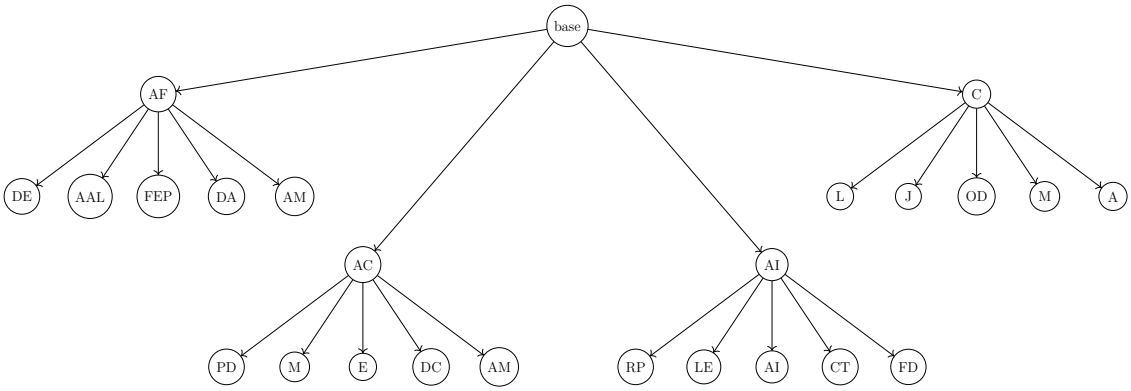


Figura 4.1. Grafo de representación del Cuestionario Dinámico sobre hobbies.

Se evalúan 3 niveles de detalle en este ejemplo. El cuestionario base (nivel 0) evalúa 4 categorías generales, a cada una de estas categorías se le asocia un cuestionario que evalúa 5 sub-categorías (nivel 1), a estas también se les asocia un cuestionario que evalúa 3 sub-sub-categorías (nivel 2). En total se evalúan 4 categorías, 20 sub-categorías y 60 sub-sub-categorías. La figura 4.1 muestra el diagrama del grafo que representa las relaciones de origen y descendencia para cada cuestionario. Se desarrolla a detalle esta estructura:

- **Cuestionario base**: Se evalúan 4 categorías generales: “Actividades físicas”, “Artes creativas”, “Actividades intelectuales” y “Coleccionismo”. Cada categoría tiene asociados 5 ítems tipo Likert. Este cuestionario cuenta con 20 ítems en total.
    - **Actividades físicas**: Se evalúan 5 sub-categorías: “Deportes de equipo”, “Actividades al aire libre”, “Fitness y entrenamiento personal”, “Deportes acuáticos” y “Artes marciales”. Cada sub-categoría tiene asociados 3 ítems tipo Likert. Este cuestionario cuenta con 15 ítems en total.
        - **Deportes de equipo**: Se evalúan 3 sub-sub-categorías: “Fútbol”, “Baloncesto” y “Voleibol”. Cada sub-sub-categoría tiene asociados 3 ítems tipo Likert. Este cuestionario cuenta con 9 ítems en total.
        - **Actividades al aire libre**: Se evalúan 3 sub-sub-categorías: “Senderismo”, “Ciclismo” y “Escalada”. Cada sub-sub-categoría tiene asociados 3 ítems tipo Likert. Este cuestionario cuenta con 9 ítems en total.
        - **Fitness y entrenamiento personal**: Se evalúan 3 sub-sub-categorías: “Yoga”, “Pilates” y “Entrenamiento de fuerza”. Cada sub-sub-categoría tiene asociados 3 ítems tipo Likert. Este cuestionario cuenta con 9 ítems en total.
        - **Deportes acuáticos**: Se evalúan 3 sub-sub-categorías: “Natación”, “Surf” y “Buceo”. Cada sub-sub-categoría tiene asociados 3 ítems tipo Likert. Este cuestionario cuenta con 9 ítems en total.
        - **Artres marciales**: Se evalúan 3 sub-sub-categorías: “Karate”, “Judo” y “Taekwondo”. Cada sub-sub-categoría tiene asociados 3 ítems tipo Likert. Este cuestionario cuenta con 9 ítems en total.
    - **Artes creativas**: Se evalúan 5 sub-categorías: “Pintura y dibujo”, “Música”, “Escritura”, “Danza y expresión corporal” y “Artesanía y manualidades”. Cada sub-categoría tiene asociados 3 ítems tipo Likert. Este cuestionario cuenta con 15 ítems en total.
        - **Pintura y dibujo**: Se evalúan 3 sub-sub-categorías: “Acuarela”, “Óleo” y “Dibujo a lápiz”. Cada sub-sub-categoría tiene asociados 3 ítems tipo Likert. Este cuestionario cuenta con 9 ítems en total.
        - **Música**: Se evalúan 3 sub-sub-categorías: “Canto”, “Piano” y “Guitarra”. Cada sub-sub-categoría tiene asociados 3 ítems tipo Likert. Este cuestionario cuenta con 9 ítems en total.

- ◇ **Escritura**: Se evalúan 3 sub-sub-categorías: “Novela”, “Poesía” y “Periodismo”. Cada sub-sub-categoría tiene asociados 3 ítems tipo Likert. Este cuestionario cuenta con 9 ítems en total.
- ◇ **Danza y expresión corporal**: Se evalúan 3 sub-sub-categorías: “Ballet”, “Bailes latinos” y “Hip hop”. Cada sub-sub-categoría tiene asociados 3 ítems tipo Likert. Este cuestionario cuenta con 9 ítems en total.
- ◇ **Artesanía y manualidades**: Se evalúan 3 sub-sub-categorías: “Cerámica”, “Tejido” y “Joyería”. Cada sub-sub-categoría tiene asociados 3 ítems tipo Likert. Este cuestionario cuenta con 9 ítems en total.

- ○ **Actividades intelectuales**: Se evalúan 5 sub-categorías: “Resolución de puzzles”, “Lectura y escritura”, “Aprendizaje de idiomas”, “Ciencia y tecnología” y “Filosofía y debate”. Cada sub-categoría tiene asociados 3 ítems tipo Likert. Este cuestionario cuenta con 15 ítems en total.
  - ◇ **Resolución de puzzles**: Se evalúan 3 sub-sub-categorías: “Rompecabezas”, “Sudokus” y “Crucigramas”. Cada sub-sub-categoría tiene asociados 3 ítems tipo Likert. Este cuestionario cuenta con 9 ítems en total.
  - ◇ **Lectura y escritura**: Se evalúan 3 sub-sub-categorías: “Novela”, “Poesía” y “Ensayo”. Cada sub-sub-categoría tiene asociados 3 ítems tipo Likert. Este cuestionario cuenta con 9 ítems en total.
  - ◇ **Aprendizaje de idiomas**: Se evalúan 3 sub-sub-categorías: “Inglés”, “Francés” y “Mandarin”. Cada sub-sub-categoría tiene asociados 3 ítems tipo Likert. Este cuestionario cuenta con 9 ítems en total.
  - ◇ **Ciencia y tecnología**: Se evalúan 3 sub-sub-categorías: “Astronomía”, “Robótica” y “Programación”. Cada sub-sub-categoría tiene asociados 3 ítems tipo Likert. Este cuestionario cuenta con 9 ítems en total.
  - ◇ **Filosofía y debate**: Se evalúan 3 sub-sub-categorías: “Ética”, “Filosofía” y “Política”. Cada sub-sub-categoría tiene asociados 3 ítems tipo Likert. Este cuestionario cuenta con 9 ítems en total.
- ○ **Coleccionismo**: Se evalúan 5 sub-categorías: “Libros”, “Juguetes”, “Objetos deportivos”, “Música” y “Automóviles”. Cada sub-categoría tiene

asociados 3 ítems tipo Likert. Este cuestionario cuenta con 15 ítems en total.

- ◇ **Libros**: Se evalúan 3 sub-sub-categorías: “Literatura clásica”, “Ciencia ficción” y “Poesía”. Cada sub-sub-categoría tiene asociados 3 ítems tipo Likert. Este cuestionario cuenta con 9 ítems en total.
- ◇ **Juguetes**: Se evalúan 3 sub-sub-categorías: “Figuras de acción”, “Muñecas” y “Juegos de mesa”. Cada sub-sub-categoría tiene asociados 3 ítems tipo Likert. Este cuestionario cuenta con 9 ítems en total.
- ◇ **Objetos deportivos**: Se evalúan 3 sub-sub-categorías: “Camisetas de equipos”, “Autógrafos” y “Trofeos”. Cada sub-sub-categoría tiene asociados 3 ítems tipo Likert. Este cuestionario cuenta con 9 ítems en total.
- ◇ **Música**: Se evalúan 3 sub-sub-categorías: “Discos de vinilo”, “Instrumentos musicales” y “Partituras”. Cada sub-sub-categoría tiene asociados 3 ítems tipo Likert. Este cuestionario cuenta con 9 ítems en total.
- ◇ **Automóviles**: Se evalúan 3 sub-sub-categorías: “Coches clásicos”, “Coches deportivos” y “Miniaturas”. Cada sub-sub-categoría tiene asociados 3 ítems tipo Likert. Este cuestionario cuenta con 9 ítems en total.

El carácter de crecimiento exponencial de los cuestionarios con la metodología de descendencia los hace particularmente grandes, en este caso se utilizaron un total de 260 ítems tipo Likert para desarrollar el Cuestionario Dinámico. Esto pondrá a prueba la eficiencia del mismo para determinar tendencias de los usuarios y concentrarse en las categorías más relevantes para cada uno de ellos.

4.2. Preparación del ejemplo

Se elaboraron los ítems para cada cuestionario en una hoja de cálculo, posteriormente estos fueron leídos utilizando la librería de Python `pandas` y se crearon las instancias de `item` para cada uno de ellos. Se crearon las instancias de `survey` para cada uno de los cuestionarios y se agregaron los ítems correspondientes a cada uno de ellos. Además se definieron funciones de probabilidad y condición distintas

en cada nivel, modificando los parámetros para ajustarse a lo que cada cuestionario requería.

4.2.1. Funciones de peso de probabilidad del ítem

Se utilizó la función de peso de probabilidad del ítem desarrollada en la ecuación 2.3 e implementada en la biblioteca en la sección 3.7.2.4. Los parámetros `not_repeated_since` y `cat_launch_count_weight` fueron definidos de forma distinta en cada nivel:

- `not_repeated_since`: Se asignó este valor igual a la cantidad de ítems de cada cuestionario menos 5, por lo que un usario no recibirá preguntas idénticas hasta que haya respondido la totalidad menos 5 de los ítems del cuestionario.
- `cat_launch_count_weight`: Se asignó este valor igual al negativo del doble de la cantidad de categorías evaluadas en dicho nivel: -8, -10, -6 respectivamente para cada nivel. Se asocia un valor negativo para reducir la aparición de ítems de una misma categoría de manera consecutiva.

4.2.2. Funciones de condición de los cuestionarios

Las condiciones de cada cuestionario fueron definidas a partir del nivel en el que se encuentran. Estas se describen a continuación:

- **Nivel 0:** Ya que este cuestionario es el origen de todos los demás, la condición para este cuestionario es constantemente `True`, la aparición de preguntas del mismo se ve drásticamente reducida cuando cuestionarios en su descendencia comienzan a cumplir la condición de lanzamiento, por a la gran cantidad de cuestionarios que existen.
- **Nivel 1:** La condición para los cuestionarios en este nivel es que el usuario haya respondido al menos 4 ítems del cuestionario base, y cumplir con la condición de threshold descrita en la ecuación 2.8, con el threshold igual a 0.5.
- **Nivel 2:** La condición para los cuestionarios en este nivel también está dada por un threshold igual a 0.5.

4.2.3. Funciones de probabilidad de los cuestionarios

La probabilidad de cada uno de los cuestionarios fue desarrollada de forma similar para cada uno de los niveles, esta se calcula de la siguiente manera:

$$P = max\left\{\left(1 - \frac{c}{d}\right), \frac{1}{d}\right\} \times R. \tag{4.1}$$

Donde:

- P es la probabilidad de lanzamiento del cuestionario.
- c es la cantidad de veces que el cuestionario ha sido respondido.
- d es la dimensión del cuestionario.
- R es un factor de ajuste que se define de forma distinta para cada nivel.

El factor de ajuste R se define de la siguiente manera para cada nivel:

- **Nivel 0:** Se define igual a 4.
- **Nivel 1:** Se define igual a 1.
- **Nivel 2:** Se define igual a 1.5.

Estas definiciones se realizaron de forma empírica, el cuestionario base es el que tiene este factor más elevado, esto es porque cuando otros cuestionarios son "desbloqueados" (cumplen con la condición y pueden ser seleccionados), la probabilidad del cuestionario base se ve drásticamente reducida por la cantidad de posibles cuestionarios a elegir. Se asignó igual a cuatro porque en caso de que todos los cuestionarios del nivel 1 sean "desbloqueados" habrán 4 cuestionarios más para seleccionar con un peso de probabilidad igual a 1 cada uno, o sea que entre los cinco cuestionarios existe 50 % de probabilidad de seleccionar el cuestionario base, y 50 % de seleccionar otro que pertenezca al nivel 1. Para el nivel 2 se asignó un valor mayor (1.5) de modo que los ítems a mostrar fueran más específicos una vez que el usuario presentara una tendencia. Se hicieron varias pruebas modificando estos valores y buscando una evolución balanceada del Cuestionario Dinámico.

4.3. Ejecución del ejemplo

Una vez definido el método para desarrollar el ejemplo, se procedió a programarlo haciendo uso de la biblioteca `pydyn_surv`. Las pruebas iniciales se realizaron

en una terminal, pero cuando este estaba listo para su uso se diseñó una interfaz gráfica utilizando un paquete de Python llamado `streamlit`. Esta interfaz permite ejecutar el ejemplo de forma sencilla, además de mostrar los resultados del modelo al usuario de forma gráfica. Una captura de la interfaz se muestra en la figura 4.2, pero se puede acceder al ejemplo completo como un usuario en el siguiente enlace: `https://pydyn-surv-hobbies-test.streamlit.app/`.

Figura 4.2. Captura de la interfaz gráfica desarrollada para el ejemplo.

Se estableció que a los usuarios se les solicitaría contestar un total de 30 ítems para esta prueba y al finalizar se les presentaba una serie de gráficas con los resultados obtenidos por el modelo. Estas gráficas mostraban las preferencias del usuario basadas en los pesos entrenados de cada nivel en el Cuestionario Dinámico, para esto se tomaban todos los cuestionarios en el nivel y los pesos asociados a estos, las categorías con pesos mayores a 0 se dividían dentro de 2 (ya que 2 es el valor máximo que una coordenada en el vector de peso puede alcanzar) y se multiplicaban por 100, este valor indicaba la preferencia porcentual de la categoría. La figura 4.4 muestra un ejemplo de los resultados obtenidos por un usuario. Se solicitó una

evaluación que consistía en ponderar el desempeño del modelo en una escala del 0 % al 100 % de certeza, además de una opción para escribir un breve comentario, como se muestra en la figura 4.3.

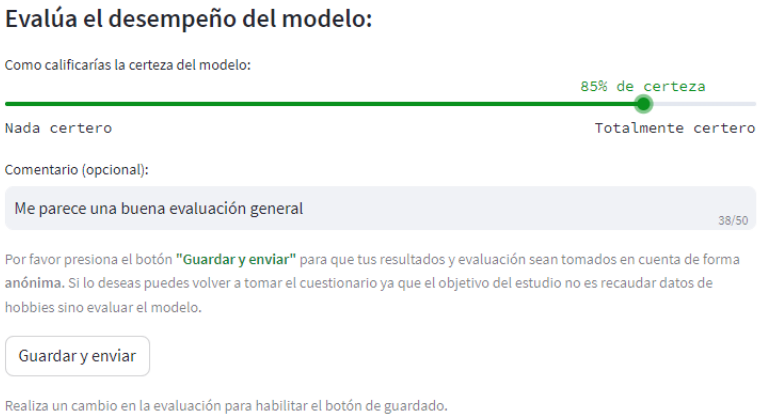
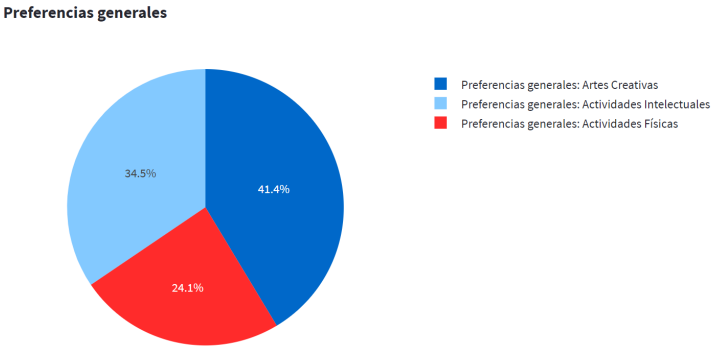
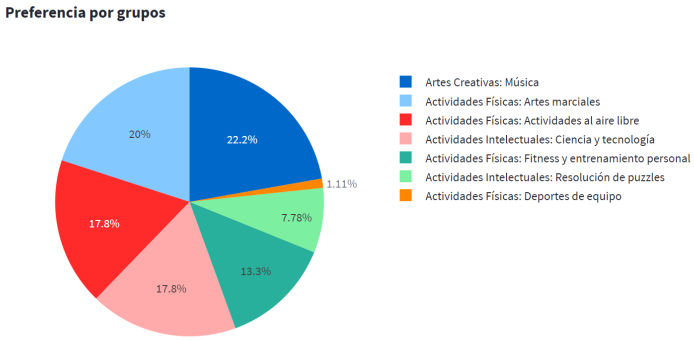


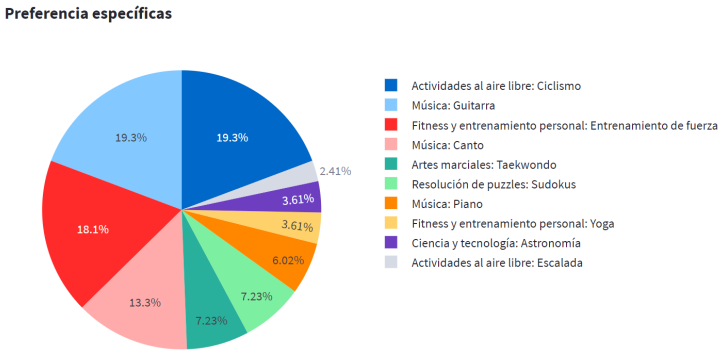
Figura 4.3. Captura de la sección de evaluación de la interfaz gráfica desarrollada para el ejemplo.



(a) Diagrama de pie que representa las preferencias del usuario en el nivel 0.



(b) Diagrama de pie que representa las preferencias del usuario en el nivel 1.



(c) Diagrama de pie que representa las preferencias del usuario en el nivel 2.

Figura 4.4. Ejemplo de los resultados obtenidos por un usuario.

4.4. Resultados del ejemplo

El ejemplo fue ejecutado por un total de 16 usuarios de forma voluntaria, las calificaciones de los usuarios promediaron un total de 72.5 % de certeza, y aunque esta calificación es subjetiva, da una idea de la percepción del modelo desde el punto de vista de los usuarios. La tabla 4.1 muestra algunos datos interesantes de cada uno de los usuarios que tomaron el Cuestionario Dinámico.

Tabla 4.1. Datos por usuario

		Valores de las respuestas					
Usuario	**Evaluación**	**Máximo**	**Mínimo**	**Promedio**	**Desviación estándar**	**Cuestionarios desbloqueados**	**Comentario**
0	70	2	-0.9	0.916667	0.787437	9	
1	55	2	-2	0.17	1.413762	9	
2	90	2	-2	0.313333	1.465402	7	Interesante cuestionario
3	90	2	-2	1.113333	1.175243	10	
4	90	2	-0.6	1.253333	0.946767	10	Predice las preferencias, es intuitivo y dinámico
5	90	1.9	-1	0.43	0.999707	7	
6	70	2	-2	0.47	1.318084	10	
7	70	2	-1.9	0.37	1.227318	9	
8	10	2	-0.9	0.163333	0.549284	3	Se repetían mucho las preguntas
9	55	2	-0.5	0.713333	0.80718	8	
10	85	2	-2	0.526667	1.419551	8	
11	85	2	-1	0.9	0.946864	9	
12	70	2	-2	1.01	1.20869	11	
13	45	2	-0.5	1.006667	0.842383	12	
14	100	2	-2	0.91	1.231862	8	
15	85	2	-1.3	0.73	0.89487	10	Me parece una buena evaluación general
Promedio	72.5	1.99375	-1.4125	0.687292	1.07715	8.75	

De esta tabla se pueden obtener algunas conclusiones:

- El usuario que calificó al modelo con menor puntuación fue el número 8, afortunadamente este dejó un comentario explicando su calificación: “Se repetían mucho las preguntas”. Al observar la columna de **cuestionarios desbloqueados** se puede observar que durante esta prueba únicamente se desbloquearon 3 cuestionarios, esto es una cantidad muy por debajo del promedio de los otros casos y explica el por qué los ítems se repitieron o eran similares. Esto se debe a que el usuario no mostró una tendencia clara en sus respuestas, por lo que el modelo no pudo determinar con certeza a qué categoría pertenecía el usuario. Esto se refuerza al notar que el valor promedio de sus respuestas fue menor al threshold de 0.5, siendo este de 0.1633 y la desviación estándar de sus respuestas también es la menor de todas, lo que indica que el usuario consistentemente respondió de forma neutral a los ítems.

- Existe una tendencia general de los usuarios a marcar con mayor confianza

porcentajes altos cuando están de acuerdo con la afirmación a marcar porcentajes altos de “en desacuerdo”.

- De los 25 cuestionarios posibles, los usuarios solo interactuaron con un promedio de 8.75 y un máximo de 12 cuestionarios, es decir el 35 % y 48 % de la totalidad de cuestionarios respectivamente. Esto habla sobre la capacidad del sistema de evolucionar y adaptarse a las preferencias de los usuarios, de lo contrario, estos hubieran interactuado con todos los cuestionarios disponibles. Además se concluye que los cuestionarios que fueron respondidos se alinean con las preferencias de los usuarios porque de no ser así estos hubieran realizado una calificación más drástica.

Para verificar que se realizaron evaluaciones en todos los niveles, se contaron los ítems que fueron respondidos en cada nivel, de 480 ítems respondidos en total, 151 pertenecían al nivel 0, 156 al nivel 1 y 173 al nivel 2, lo que indica que los usuarios interactuaron con todos los niveles de forma equitativa con una mayor tendencia hacia los niveles más específicos. Porcentualmente cada nivel fue respondido en un 31.46 %, 32.5 % y 36.04 % respectivamente como lo muestra la tabla 4.2.

Tabla 4.2. Porcentaje de ítems respondidos por nivel.

Nivel	**Porcentaje**
0	31.46 %
1	32.5 %
2	36.04 %

Con los datos registrados es posible hacer un análisis al rededor de las preferencias en los pasatiempos de los usuarios que participaron del ejemplo. Sin embargo ese no es el objetivo de este trabajo, por lo que se omitirá este análisis.

Un aprendizaje importante de este ejemplo fue que el uso de una cantidad tan elevada de cuestionarios puede ser contraproducente, ya que cada uno de estos se entrena por separado y distribuir las respuestas de los usuarios en varios cuestionarios reduce la cantidad de datos de entrenamiento para cada uno. Debido al alto número de cuestionarios y la cantidad reducida de iteraciones de cada uno, realizar análisis de evolución como los expuestos en la sección 2.3 no fue viable. En futuras investigaciones o usos del modelo, se recomienda definir una cantidad menor de cuestionarios, que contengan en sí mismos los distintos niveles de precisión.

CONCLUSIONES

1. La ciencia de la complejidad busca comprender la emergencia de fenómenos en los sistemas complejos causados por las interacciones de sus agentes. Se consiguió documentar acerca de estos sistemas, exponiendo su definición, características, métodos de estudio y aplicaciones.

   Por otro lado, el machine learning es un sub-dominio de la inteligencia artificial que busca que las computadoras aprendan de los datos sin ser programadas explícitamente. Se consiguió documentar acerca del mismo, exponiendo las definiciones y procedimientos más comunes, además de las aplicaciones más populares.

   Existe una amplia relación entre estos campos, tanto en el uso de machine learning para el estudio de los sistemas complejos, como el uso de los sistemas complejos en el diseño de modelos de machine learning.

2. Se desarrolló un modelo funcional, que calcula la evolución de un cuestionario dinámico adaptándose a las respuestas emitidas por un usuario. Se observó que prioriza aquellos ítems que se alinean con los intereses de los usuarios, y determina de manera adecuada el orden de preferencia de las categorías evaluadas. Además, se desarrolló una metodología que considera múltiples cuestionarios interrelacionados.

3. El modelo se implementó en una biblioteca de Python, que permite la creación de cuestionarios dinámicos utilizando esta metodología. Esta biblioteca permite una fácil implementación del modelo que puede adaptarse a múltiples necesidades y proyectos debido a la versatilidad del mismo.

   Dentro de los ejemplos incluidos en la biblioteca se muestra como obtener los ítems y sus vectores a partir de una hoja de cálculo, además de dos implementaciones de interfaz de usuario sencillas y minimalistas, una desarrollada para ser ejecutada en una terminal, y la otra con un sistema gráfico mostrado

en un navegador. También se incluye la documentación de uso de la misma, exponiendo todas las funciones, clases y métodos de los que está compuesta.

La biblioteca se encuentra de forma libre en `https://github.com/JARA99/Dynamic_Survey_lib`.

4. El ejemplo de aplicación expone las capacidades de entrenamiento y predicción del modelo, el cual al ser desarrollado con múltiples cuestionarios, evoluciona añadiendo detalle a las categorías de interés de sus usuarios. Esto sugiere que investigaciones donde se amplíen elementos del modelo pueden ser relevantes. Se resalta la exploración en la inclusión de ítems distintos al de Likert y también un refinamiento de la activación de cuestionarios en el caso donde haya un alto volumen de estos. El cuestionario del ejemplo se encuentra disponible en `https://pydyn-surv-hobbies-test.streamlit.app/`.

RECOMENDACIONES

1. Debido a las características de complejidad del modelo, se recomienda realizar un estudio más profundo sobre las posibles propiedades emergentes que surjan del mismo.

2. El uso de la metodología de cuestionarios múltiples expuesta en la sección 2.2.5 debe ser realizado con cautela. Puesto que en esta metodología el entrenamiento de cada cuestionario se realiza de forma independiente, aumentar la cantidad de cuestionarios se transforma en sacrificar la cantidad de elementos en el training data set de cada uno. En ese sentido, se recomiendan las siguientes acciones:

    - Definir un único cuestionario principal que evalúe todas las categorías y sub categorías en sí mismo y utilizar la metodología de descendencia únicamente para definir grupos de ítems de los que se requiera limitar o censurar la aparición, a menos que se cumplan ciertas condiciones.
    - Desarrollar una mejora al sistema, que permita que los ítems contestados en un cuestionario descendiente de otro agreguen información al training data set del cuestionario origen.

3. Se recomienda utilizar un deslizador de selección en el diseño de la interfaz de respuesta del cuestionario. Esto permitirá que el usuario pueda seleccionar valores intermedios entre las opciones de respuesta, aumentando la precisión de las etiquetas en el training data set.

4. Para futuras investigaciones, se recomienda diseñar sistemas que permitan el uso de otros tipos de ítem a parte de los de Likert.

5. Se recomienda reducir la cantidad de iteraciones de entrenamiento del modelo progresivamente. Ya que cada entrenamiento solo reajusta el peso tomando en cuenta el nuevo ítem agregado, no es necesario realizar tantas iteraciones cuando el modelo ya ha sido entrenado con una gran cantidad de ítems.

BIBLIOGRAFÍA

[1] Anderson, P. W. Is Complexity Physics? Is It Science? What is It? *Physics Today 44*, 7 (07 1991), 9–11.

[2] Arribas, M. Diseño y validación de cuestionarios. *Matronas profesión 5*, 17 (2004), 23–29.

[3] Blundell, S., and Blundell, K. *Concepts in Thermal Physics*. OUP Oxford, 2009.

[4] Carleo, G., Cirac, I., Cranmer, K., Daudet, L., Schuld, M., Tishby, N., Vogt-Maranto, L., and Zdeborová, L. Machine learning and the physical sciences. *Reviews of Modern Physics 91*, 4 (dec 2019).

[5] Castagna, R., and Bigelow, S. J. Definition: information technology (it). `https://www.techtarget.com/searchdatacenter/definition/IT`, agosto 2021. (Accedido el 08/10/2023).

[6] Chacon, S. Git. `https://git-scm.com/`. (Accedido el 06/08/2023).

[7] Chicco, D., Warrens, M. J., and Jurman, G. The coefficient of determination r-squared is more informative than smape, mae, mape, mse and rmse in regression analysis evaluation. *PeerJ Computer Science 7* (2021), e623.

[8] Doherty, E. What is object-oriented programming? oop explained in depth. `https://www.educative.io/blog/object-oriented-programming`, abril 2020. (Accedido el 07/27/2023).

[9] Esposito, M., Bheemaiah, K., and Tse, T. What is machine learning? `https://theconversation.com/what-is-machine-learning-76759`, mayo 2017. (Accedido el 07/16/2023).

[10] Ganssle, P. Why you shouldn't invoke setup.py directly. https://blog.ganssle.io/articles/2021/10/setup-py-deprecated.html, octubre 2021. (Accedido el 06/06/2023).

[11] García, R. Interdisciplinariedad y sistemas complejos. *Revista Latinoamericana de Metodología de las Ciencias Sociales: Relmecs 1*, 1 (2011), 66–101.

[12] Google for Developers. Framing: Key ml terminology. https://developers.google.com/machine-learning/crash-course/framing/ml-terminology, julio 2022. (Accessed on 07/20/2023).

[13] Gross, J. L., and Yellen, J. *Handbook of graph theory*. CRC press, 2003.

[14] Hahs-Vaughn, D. L., and Lomax, R. G. *An introduction to statistical concepts*. Routledge, 2020.

[15] Israel, G. The science of complexity: Epistemological problems and perspectives. *Science in Context 18*, 3 (2005), 479–509.

[16] ivan_pozdeev. Python - setuptools vs. distutils: why is distutils still a thing? - stack overflow. https://stackoverflow.com/questions/25337706/setuptools-vs-distutils-why-is-distutils-still-a-thing, octubre 2016. (Accedido el 06/06/2023).

[17] Jiménez, J. C., Sánchez, J. G., and Aguilar, F. G. Guía técnica para la construcción de cuestionarios. *Odisea Revista electrónica de pedagogía 3*, 6 (2006).

[18] Kaviani, S., and Sohn, I. Application of complex systems topologies in artificial neural networks optimization: An overview. *Expert Systems with Applications 180* (2021), 115073.

[19] Kollár, L. K. Managing python packages the right way. https://opensource.com/article/19/4/managing-python-packages, agosto 2020. (Accedido el 06/10/2023).

[20] Kwame, A. E., Martey, E. M., and Chris, A. G. Qualitative assessment of compiled, interpreted and hybrid programming languages. *Communications 7*, 7 (2017), 8–13.

[21] Lardinois, F., and Lunden, I. Microsoft has acquired github for $7.5b in stock | techcrunch. https://techcrunch.com/2018/06/04/microsoft-has-acquired-github-for-7-5b-in-microsoft-stock/, junio 2018. (Accedido el 06/10/2023).

[22] Levine, J. *Linkers and Loaders*. Operating Systems Series. Elsevier Science, 2000.

[23] Liang, P. Machine learning 1 - linear classifiers, sgd | stanford cs221: Ai (autumn 2019) - youtube. https://www.youtube.com/watch?v=zrT2qETJilw. (Accedido el 01/02/2023).

[24] Luna, S. M. M. Manual práctico para el diseño de la escala likert. *Xihmai 2*, 4 (2007).

[25] Lutz, M. *Learning python: Powerful object-oriented programming*. O'Reilly Media, Inc., 2013.

[26] McGranaghan, R. M. Complexity heliophysics: A lived and living history of systems and complexity science in heliophysics. *arXiv preprint arXiv:2307.03287* (2023).

[27] Mohri, M., Rostamizadeh, A., and Talwalkar, A. *Foundations of machine learning*. MIT press, 2018.

[28] Murphy, K. A., and Bassett, D. S. Information decomposition to identify relevant variation in complex systems with machine learning. *arXiv preprint arXiv:2307.04755* (2023).

[29] Newman, M. E. J. Resource Letter CS–1: Complex Systems. *American Journal of Physics 79*, 8 (08 2011), 800–810.

[30] Petter, S., DeLone, W., and McLean, E. Measuring information systems success: models, dimensions, measures, and interrelationships. *European journal of information systems 17* (2008), 236–263.

[31] Phelan, S. E. What is complexity science, really? *Emergence 3*, 1 (2001), 120–136.

[32] Preston-Werner, T. Semantic versioning 2.0.0. https://semver.org/, junio 2013. (Accedido el 06/07/2023).

[33] Python Packaging Authority. Building and distributing packages with setuptools - setuptools 67.8.0.post20230529 documentation. https://setuptools.pypa.io/en/latest/setuptools.html, mayo 2023. (Accedido el 06/06/2023).

[34] Python Packaging Authority. Quickstart - setuptools 67.8.0.post20230529 documentation. https://setuptools.pypa.io/en/latest/userguide/quickstart.html#setuppy-discouraged, mayo 2023. (Accedido el 06/06/2023).

[35] Python Software Foundation. 6. modules — python 3.11.3 documentation. https://docs.python.org/3/tutorial/modules.html, 2001. (Accedido el 06/04/2023).

[36] QuestionPro. Developing dynamic surveys. https://www.surveyanalytics.com/images/bookshelf/dynamicsurveys.pdf, 2007. (Accedido el 08/10/2023).

[37] Real Academia Española. *Diccionario de la lengua española*, 23 ed. 2023. [versión 23.6 en línea].

[38] Ricci, F., Rokach, L., and Shapira, B. *Recommender Systems: Techniques, Applications, and Challenges*. Springer US, New York, NY, 2022, pp. 1–35.

[39] Rosas, F. E., Mediano, P. A., Jensen, H. J., Seth, A. K., Barrett, A. B., Carhart-Harris, R. L., and Bor, D. Reconciling emergences: An information-theoretic approach to identify causal emergence in multivariate data. *PLoS computational biology 16*, 12 (2020), e1008289.

[40] Ruder, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2017).

[41] Rudin, C. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature machine intelligence 1*, 5 (2019), 206–215.

[42] Trudeau, R. J. *Introduction to graph theory*. Courier Corporation, 2013.

[43] Van Rossum, G., Drake, F. L., et al. *Python Reference Manual*, 2.1.1 ed. Stichting Mathematisch Centrum Amsterdam, The Netherlands., julio 2001.

[44] Vicsek, T. Complexity: The bigger picture. *Nature 418*, 6894 (2002), 131–131.