Proyecto de Simulación e Inteligencia Artificial (SIA)

Autores:

Marco Antonio Ochil Trujillo

Kevin Majim Ortega Alvarez

Problema: Evaluación de un sistema de recomendación de películas simulando la reacción de los usuarios

Introducción:

En este proyecto, nos propusimos mejorar la efectividad de los sistemas de recomendación a través de la simulación de la interacción de usuarios y sistemas. Para lograr este objetivo, hemos diseñado un enfoque integral que combina temas de búsqueda, conocimiento y procesamiento de lenguaje natural.

El Problema:

Se tiene un sitio web de películas, al cual el usuario, le ingresa a partir de una selección, que películas le han gustado y cuáles no, además se le permite añadir comentarios sobre las películas y en base a esos gustos, devuelve otro grupo de películas que recomienda. Dado la salida de ese sistema, se desea simular la reacción del usuario para determinar, cuales películas pueden gustarle y cuales no, para devolver esa reacción al sistema de recomendación como base para su retroalimentación, y para evaluar, que tan bueno es ese sistema de recomendación, puesto que las reacciones de los usuarios son cruciales, para mejorar las recomendaciones dadas por el sistema, y ofrecer una experiencia personalizada y satisfactoria.

Para resolver este problema contamos con una base de datos, que contiene la información de las películas, así como otras dos, que poseen relevancia de los géneros en las películas y opiniones positivas y negativas de usuarios a ellas.

Interfaz del sitio:

En la página inicial (Imagen 1) se muestran 30 películas escogidas al azar en las que el usuario puede votar positiva o negativamente así como comentar; además para el caso de querer opinar de una película en específico se tiene la opción del buscador.

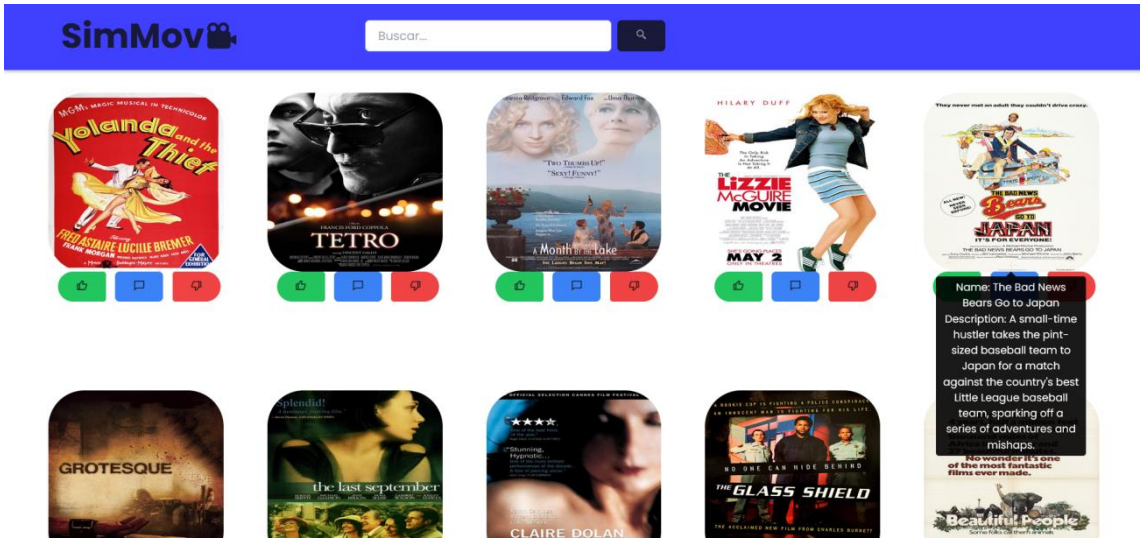

Imagen 1: Página Inicial

En caso de desearse más información sobre una determinada película, cada una cuenta con una interfaz que muestra más datos sobre la misma (Imagen 2), solo se debe hacer click sobre su poster.

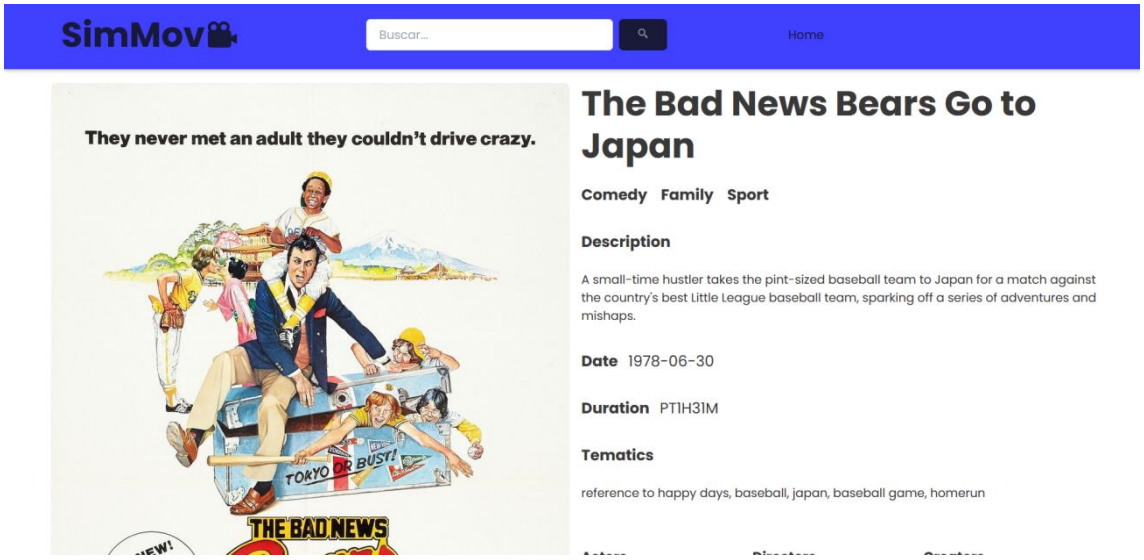

Imagen. 2: Página asociada a "The Bad News Bears Go to Japan"

Técnicas utilizadas:

Sistemas de Recomendación:

El objetivo era crear un sistema de recomendación básico que se encarga de obtener la entrada del usuario y devolverle una serie de películas basadas en sus gustos. Al inicio teníamos un sistema con un modelo que calculaba la distancia de Jaccard [1]; dado que al ser impartido en el curso de Sistemas de Recuperación de Información este semestre, poseíamos bastante conocimiento del mismo, además de ser un modelo sencillo, pero muy efectivo, por lo que su implementación se nos facilitó, para realizar las primeras pruebas. Al determinar la distancia de Jaccard con los usuarios de la base de datos en "users.json" se organizaban (del usuario más cercano al más lejano) y se tomaban a lo sumo 3 películas positivamente valoradas de cada usuario hasta completar 10 sugerencias.

Luego del sistema de Jaccard decidimos hacer recomendaciones, con algoritmos genéticos (investigamos que se había hecho [2]) y distancia de Jaccard. Con lo que se formó un algoritmo un poco más complejo, que consta de los siguientes pasos:

1- Se determina la distancia de Jaccard y se organizan los usuarios.
2- Se toman todas las películas positivamente valoradas por los usuarios con una cercanía superior al umbral establecido (0.2, tomado de experimentación). Para situaciones donde dicho umbral presenta pocos ejemplares están determinados ciertos procedimientos para disminuir el umbral y obtener una cantidad de ejemplares suficiente.

3- Con las opiniones de todos los usuarios sobre las películas seleccionadas se forma un modelo de Mínimos Cuadrados Alternantes [3] (más conocido por sus siglas en inglés ALS).
4- Luego se genera una población inicial de 1000 muestras donde cada una contiene 10 elementos de las películas seleccionadas y se realiza con estas un algoritmo genético de 20 generaciones, donde la evaluación se realiza utilizando el modelo ALS (Imagen 3). En este existe una probabilidad de mutación de 0.01 para cada elemento de cada muestra, donde al mutar se convierte en otro elemento del conjunto de películas que se obtuvieron en el paso 2.
5- Al terminar el proceso se evalúan y comparan los elementos de la población final y se devuelve el mejor ejemplar, para explicación más concisa del proceso: observar Imagen 4.

```python
def evaluate(ids_movies):
    quality = 0

    for i, id_movie in enumerate(ids_movies):
        represent = als_model.item_factors[id_movie]
        sim = np.dot(represent, als_model.user_factors.T)
        quality_mov = np.sum(sim * matrix[:, i])
        quality += quality_mov

    return quality
```

Imagen 3: Función de evaluación de una muestra poblacional en el algoritmo genético

```python
poblation = init_pob()

for _ in range(num_generations):
    new_poblation = []
    for _ in range(size_poblation):
        father1 = select(poblation)
        father2 = select(poblation)
        son = crossing(father1, father2)
        son = mutate(son)
        new_poblation.append(son)

    poblation = new_poblation

best = set(max(poblation, key=evaluate))
```

Imagen 4: Proceso del algoritmo genético

Hay que tener en cuenta que 2 padres que dan lugar a un elemento de la generación siguiente pueden tener películas en común, dando lugar a repeticiones en generaciones siguientes cuando se realiza el cruzamiento, con esto en mente consideramos que el aumento de generaciones puede reducir el número de recomendaciones al final del algoritmo. Por otro lado el número de muestras de la población puede aportar mayor variabilidad y contrarrestar el suceso anterior, por lo que decidimos hacer distintas pruebas variando ambos parámetros.

Como ya se mencionó en el sistema con algoritmo genético desarrollado se genera un modelo ALS utilizando la opinión de todos los usuario sobre las películas seleccionadas para complementar opiniones cercanas con la opinión general de los usuarios, pero también desarrollamos otro sistemas donde se varia este aspecto y solo se utilizan los criterios de usuarios cercanos sobre las películas seleccionadas.

Con las distintas variaciones mencionadas se crearon diversos sistemas de recomendación que fueron comparados con la simulación desarrollada en base al porcentaje de resultados positivos que ofrecen (cantidad de recomendaciones que, según la simulación, fueron bien recibidas por el usuario). Los resultados de estos experimentos serán mostrados en su sección correspondiente.

Simulación:

Existe un agente principal y un conjunto de agentes de modelación BDI que reciben datos del usuario y partes de la base de datos para simular la reacción del usuario. El agente principal es el encargado de inicializar las creencias de los agentes que se pueden utilizar dada la información proporcionada por el usuario. Este agente también cumple la función de recibir la recomendación generada por el sistema y llevarla a los agentes inicializados, para posteriormente decidir un criterio de aceptación o rechazo para cada película utilizando el criterio de los agentes utilizados en el proceso. Este agente se encuentra en “cent_agent.py”.

Tenemos un agente (ubicado en “vect_agent.py”) que genera sus creencias iniciales utilizando el criterio (positivo o negativo) del usuario sobre las películas y la base de datos de “genome.json” (Imagen 5) (donde se encuentra un porcentaje de relevancia de determinados géneros para cada película). Cada película que aparece en este archivo tiene relacionado un vector de 1128 dimensiones de float entre 0 y 1; cuando uno de estos vectores va a ser incorporado a las creencias se compara con los que ya existen de su mismo criterio y en caso de tener una similitud mayor que 0.7 (según la similitud del coseno) se recalcula dicha creencia incorporando el nuevo vector, para el caso donde no existe ninguna creencia lo suficientemente cercana se toma este vector como una nueva creencia. Con esto conseguimos distintas creencias formadas por agrupación de gustos similares, luego cuando se recibe una recomendación se compara cada película con las creencias existentes (tanto positivas como negativas) utilizando la similitud del coseno, devolviendo el valor de la mayor cercanía encontrada para cada elemento, en caso de encontrarse en las creencias negativas, el valor se devuelve multiplicado por -1.

```python
def __init__(self, liked, genome):
    self.likes = []
    self.dislikes = []
    for movie in liked:
        if not movie in genome: continue
        if liked[movie] == 1: add_genoma(self.likes, genome[movie])
        else: add_genoma(self.dislikes, genome[movie])
```

Imagen 5: Inicialización de Agente Vectorial

Otro agente que utilizamos para la reacción del usuario es un agente que como base para sus creencias y su reacción utiliza lógica difusa (ubicado en “diffuse.py”), simulando un sistema experto. Este agente en su base de creencias posee actores, descripciones y directores de las películas elegidas por el usuario como sus gustos, además de varios chats con un modelo de lenguaje Gemini [4], al que se le envía la información que posee en su base de creencias para saber que tanto puede gustar la película que está recomendando el sistema. Para enviar dicha información se crean tres chats distintos con el modelo para mantener las conversaciones de cada tema separadas. En la imagen 6 se muestra como se inicializa un chat para enviar descripciones, de la misma manera se inicializan el resto de chats.

```python
def _descrip(self):
    chat=model.start_chat(history=[])
    text='I am going to send you several descriptions in the following format:\n'
    text+='description: [description1]\n'
    text+='description: [description2]\n ...\n'
    text+='and I need you to respond:'
    text+='How much could I like the \"new_description\" given that you liked the previous ones?\n'
    text+='new_description: [new description]'
    response=self._send_(chat,text)
    print(response.text)
    return chat
```

Imagen. 6: Inicializar un chat

Una vez obtenida la respuesta de Gemini, es necesario que esa respuesta sea formateada para encontrar solo los valores de interés, en la imagen 7 tenemos el método encargado de esta tarea. Para variar el proceso de experimentación también se utilizó el modelo mistral Q5_K_M integrado en LM Studio, que fue la primera herramienta utilizada para esta tarea en nuestro proyecto, donde el proceso de formato es más complejo.

```python
def _parse_(self,response,chat,text):
    parse=False

    while not parse:
        try:
            try:
                value = int(response.text)
                parse = True
            except:
                try:
                    escaped=re.escape('**')
                    splited=re.split(f'[{escaped}]',response.text)
                    value=int(splited[2])
                    parse=True
                except:
                    escaped=re.escape('[]')
                    splited=re.split(f'[{escaped}]',response.text)
                    if len(splited[1]) > 2:
                        escaped = re.escape('/')
                        splited=re.split(f'[{escaped}]',splited[1])
                    value=int(splited[1] if len(splited)>2 else splited[0])
                    parse=True
        except:
            response=self._send_(chat,text)

    return value
```

Imagen. 7: Método de formato

Entonces una vez obtenida la comparación (un valor de 0 a 10), utilizamos inferencias en lógica difusa para verificar si una película es de su gusto.

El sistema difuso, está compuesto por las siguientes variables de entrada, las cuales son las más relevantes en el momento de investigar una película:

- Actores
- Descripción
- Director

Las tres variables de entradas están representadas con funciones de membresía triangular dado que tienen un grado de pertenencia regular y como variable de salida el gusto que puede provocar esa película en el usuario, la cual está representada por 5 conjuntos (muy poco, poco, moderado, decente y mucho) y cada conjunto posee forma trapezoidal.

Dado que lo que se desea es formar un sistema experto con esta lógica difusa, investigamos sobre las variables más relevantes y encontramos que en [5] utilizaban las variables que estamos analizando en nuestro problema. Como todo sistema de lógica difusa, necesita reglas de inferencia, las cuales son proporcionadas por un experto (en el caso de [5] un cinéfilo). Antes de encontrar este artículo, propusimos demasiadas reglas para la inferencia, las cuales pudieron ser redundantes en algunos aspectos y daban resultados poco coherentes, luego filtramos esas reglas y creamos otras (Imagen 8). En el caso de las primeras reglas, son las básicas en las que si todas las categorías obtienen la misma clasificación entonces el gusto dependerá de esa clasificación. El resto de reglas son combinaciones un poco más específicas donde a criterio del cinéfilo se le dio más peso al director y a los actores, por tanto, reglas como la 10 dan baja calificación y otras como la 11 dan buena calificación.

Si el valor defuzzificado se encuentra en el rango de “lot” (Imagen 9) la película recomendada entonces se interpreta como aceptada por el usuario. En caso de no cumplirse lo anterior, pero el gusto por la película supera un umbral (Imagen 10) entonces esa película también es aceptada. Estos criterios de aceptación se diferencian con el objetivo de establecer diferentes comportamientos de recalculo de creencias, aspecto del que hablaremos en secciones próximas. Para el proceso de defuzzificación utilizamos el método del centroide.

```
rule1 = ctrl.Rule(description['poor'] & actor['poor'] & director['poor'], like['poor'])
rule2 = ctrl.Rule(description['poor'] & actor['poor'] & director['average'], like['poor'])
rule3 = ctrl.Rule(description['good'] & actor['good'] & director['good'], like['lot'])
rule4 = ctrl.Rule(description['good'] & actor['good'] & director['average'], like['lot'])
rule5 = ctrl.Rule(description['average'] & actor['average'] & director['average'], like['medium'])
rule7= ctrl.Rule(actor['good'] & (description['average'] | director['average']) |
                 description['good'] & (director['average'] | actor['average']),like['decent'])
rule8 = ctrl.Rule(director['good'] & actor['good'] & description['average'],like['decent'])
rule9 = ctrl.Rule(director['good'] & actor['poor'] & description['poor'],like['little'])
rule10 = ctrl.Rule(actor['poor'] & (description['average'] | director['average']) |
                 description['poor'] & (director['average'] | actor['average']),like['little'])
rule11 = ctrl.Rule((description['average'] & (actor['good'] | director['good'])) |
                  (actor['average'] & (description['good'] | director['good'])) |
                  (director['average'] & (description['good'] | actor['good'])), like['decent'])
rule12 = ctrl.Rule((description['average'] & (actor['poor'] | director['poor'])) |
                  (actor['average'] & (description['poor'] | director['poor'])) |
                  (director['average'] & (description['poor'] | actor['poor'])), like['little'])
```

Imagen. 8: Reglas del sistema difuso

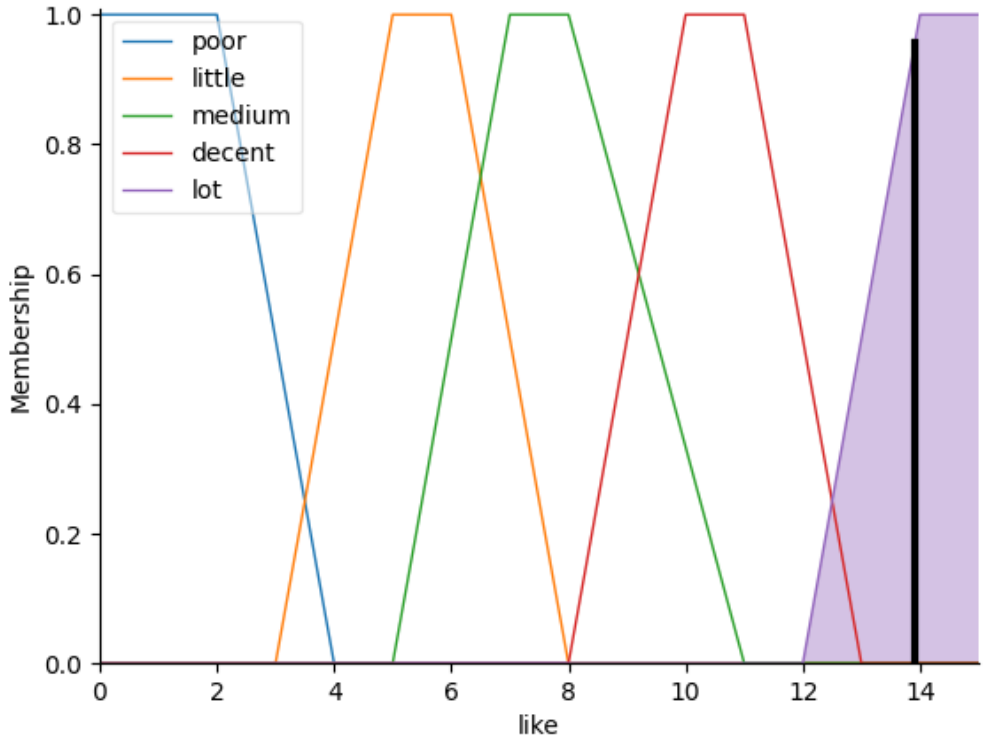

Imagen. 9: Ejemplo de aceptación 1

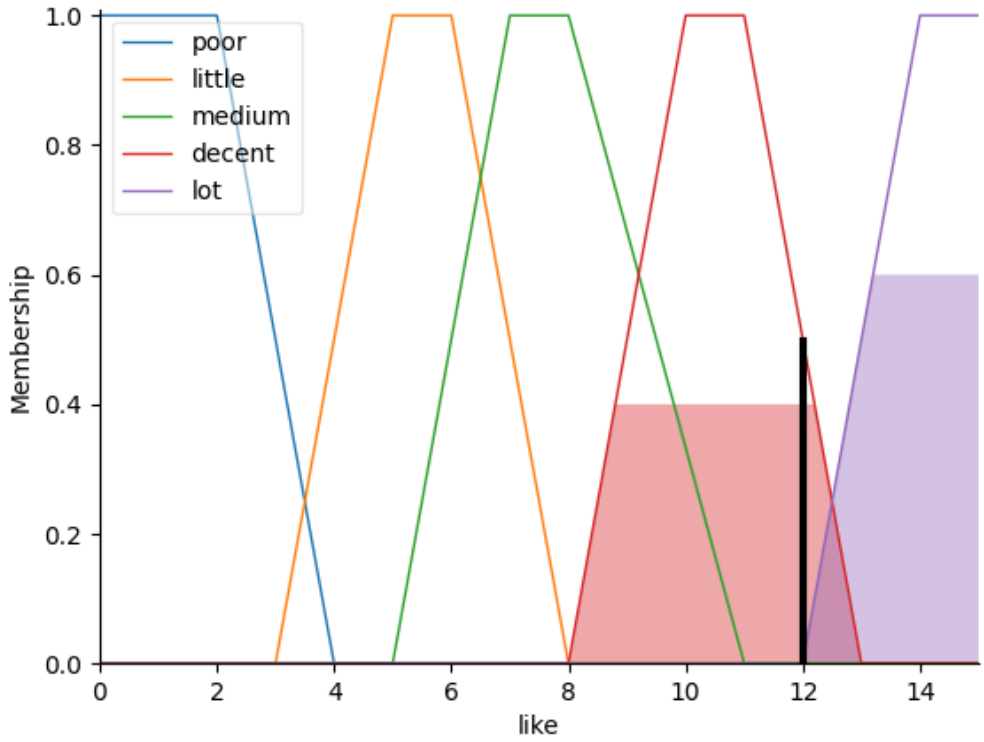

Imagen. 10: Ejemplo de aceptación 2

Existe otro agente (ubicado en “com_agent.py”), el cual trabaja con los comentarios insertados por el usuario, así como con las descripciones, palabras clave y reseñas de otros usuarios asociadas a las películas en la base de datos “movie.json”. Para generar las creencias utiliza la clase SentimentIntensityAnalyzer de nltk.sentiment, la cual es comúnmente utilizada para calcular la polaridad de un texto, en este caso solo evaluamos la polaridad del comentario del usuario (Imagen 11). Luego al recibir las películas recomendadas por el sistema se utiliza un modelo pre-entrenado de la familia MiniLM (que es una familia de versiones más pequeñas y eficientes del modelo de lenguaje BERT) para comparar las descripciones, palabras clave y reseñas de usuarios de las películas recomendadas con las que se encuentran en las creencias del agente. Para esto se utilizan los embeddings generados por MiniLM, que luego se comparan con la similitud del coseno (Imagen 12) para determinar cuál de las creencias se acerca más a la recomendación. Teniendo la polaridad del comentario y la similitud de la película, se devuelve la multiplicación de los valores como criterio final.

```python
def __init__(self, comments: dict, movies: dict):
    sia = SentimentIntensityAnalyzer()
    self.desc, self.kw, self.rev, self.beliefs = {}, {}, {}, {}
    for c in comments:
        if movies[c]['description']: self.desc[c] = movies[c]['description']
        if movies[c]['keywords']: self.kw[c] = movies[c]['keywords']
        if movies[c]['review']['reviewBody']: self.rev[c] = movies[c]['review']['reviewBody']
        self.beliefs[c] = sia.polarity_scores(comments[c])['compound']
```

Imagen. 11: Inicialización de Agente de Comentarios

```python
def similarity(self, model, text1, text2):
    emb_text1 = model.encode([text1], convert_to_tensor=True).mean(dim=0)
    emb_text2 = model.encode([text2], convert_to_tensor=True).mean(dim=0)
    return 1 - cosine(emb_text1, emb_text2)
```

Imagen. 12: Similitud de embeddings

Resultados:

Para la evaluación de los distintos sistemas de recomendación se utilizaron los criterios de los últimos 5 usuarios de la base de datos, eliminándolos del proceso de recomendación. Se tomó aproximadamente el 60% de sus opiniones para formar distintas muestras que se utilizaron como información inicial para los sistemas de recomendación y los agentes en distintos experimentos. Con esto no se podía utilizar el Agente de Comentarios por lo que se hicieron más experimentos en los que la probabilidad de incluir un comentario afín a la opinión del usuario (comentario positivo asociado a opinión positiva, de igual manera con comentarios negativos) era de 0.2. Estos comentarios fueron generados por nosotros por lo que presentan una experimentación de menor confiabilidad pero que permite observar el impacto de este agente en la simulación.

Leyenda:
Gen: Genético
Pob: Población
Com: Comentarios
Simp: Simple (haciendo alusión al algoritmo genético que utiliza menos usuarios para formar la matriz ALS)

Simulador	Cantidad de Gen	Tamaño de Pob	LLM	Simulación sin Com	Simulación con Com	Final
Jaccard	----------	----------	Gemini	40%	19.3%	29.6%
Gen Simp	5	100	Gemini	42.8%	16%	29.2%
Gen Simp	5	500	Gemini	----------	----------	----------
Gen Simp	5	1000	Gemini	----------	----------	----------
Gen Simp	10	100	Gemini	----------	----------	----------
Gen Simp	10	500	Gemini	42.8%	20%	31.5%
Gen Simp	10	1000	Gemini	----------	----------	----------
Gen Simp	20	100	Gemini	----------	----------	----------
Gen Simp	20	500	Gemini	----------	----------	----------
Gen Simp	20	1000	Gemini	----------	----------	----------
Gen Total	5	100	Gemini	31%	20%	28.2%
Gen Total	5	500	Gemini	34.7%	----------	34.7%
Gen Total	5	1000	Gemini	58.6%	----------	58.6%
Gen Total	10	100	Gemini	36%	22.2%	32.5%
Gen Total	10	500	Gemini	53.5%	----------	53.5%
Gen Total	10	1000	Gemini	40.7%	----------	40.7%

Gen Total	20	100	Gemini	25%	----------	25%
Gen Total	20	500	Gemini	37%	----------	37%
Gen Total	20	1000	Gemini	42.8%	----------	42.8%
Jaccard	----------	----------	Mistral	31.3%	18.6%	25%
Gen Simp	5	100	Mistral	----------	----------	----------
Gen Simp	5	500	Mistral	----------	----------	----------
Gen Simp	5	1000	Mistral	----------	----------	----------
Gen Simp	10	100	Mistral	----------	----------	----------
Gen Simp	10	500	Mistral	----------	----------	----------
Gen Simp	10	1000	Mistral	----------	----------	----------
Gen Simp	20	100	Mistral	----------	----------	----------
Gen Simp	20	500	Mistral	----------	----------	----------
Gen Simp	20	1000	Mistral	----------	----------	----------
Gen Total	5	100	Mistral	48.2%	----------	48.2%
Gen Total	5	500	Mistral	46.4%	----------	46.4%
Gen Total	5	1000	Mistral	21%	----------	21%
Gen Total	10	100	Mistral	42.3%	----------	42.3%
Gen Total	10	500	Mistral	32.1%	----------	32.1%
Gen Total	10	1000	Mistral	35.2%	----------	35.2%
Gen Total	20	100	Mistral	50%	----------	50%
Gen Total	20	500	Mistral	38.4%	----------	38.4%
Gen Total	20	1000	Mistral	23.5%	----------	23.5%

Como ya se mencionó para estos datos se tomó aproximadamente el 60% de las opiniones de estos usuarios, esto se hizo con el objetivo de dejar fuera de la información utilizada en el proceso de recomendación a elementos que pudieran coincidir con los recomendados, con lo que, conociendo la opinión real del usuario así como la salida final de la simulación y de cada uno de sus agentes se pudiera determinar la veracidad de la simulación y los aspectos a mejorar de la misma, utilizando las métricas de clasificación aprendidas en la asignatura de Sistemas de Recuperación de Información. Para el caso de los Agentes Vectorial y de Comentarios (los cuales no devuelven criterios de aceptación y rechazo) se estableció un umbral de 0.75 y 0.23 respectivamente (basado en experimentación) donde solo las evaluaciones superiores son aceptadas para la obtención de estos datos

Utilizando Gemini:

Agente Vectorial

105	3
3	11

Agente Difuso

32	2
4	84

Agente de Comentarios

12	1
3	37

Agente Principal

44	2
4	72

Utilizando Mistral:

Agente Vectorial

55	2
2	3

Agente Difuso

8	0
4	50

Agente de Comentarios

5	0
3	14

Agente Principal

15	0
4	43

Continuación del proyecto:

En la continuación quisiéramos:

- Incluir criterios de recalculo de creencias que fueron trabajados en los Agentes Vectorial y Difuso con el objetivo de aprender y adaptarse a los gustos del usuario en varias etapas de recomendación, así como crear criterios de recalculo de creencias para el Agente de Comentarios.

- Utilizar los datos de los experimentos mostrados para recalcular los pesos utilizados dentro de los agentes en sus funciones de aceptación o en su integración al Agente Principal, así como incluir mayor peso a los criterios de rechazo de los agentes, los cuales se trabajaron con menor relevancia que los criterios de aceptación.

- Existe información extraída de la interacción del usuario con el sitio web que no llegó a utilizarse y nos gustaría crear agentes donde pudiera ser de utilidad.

- Se creó un agente que trabajaba con la similitud entre imágenes (en este caso se utiliza para los posters) pero no llego a incluirse en el proyecto por dificultades en su experimentación a la hora de encontrar criterios de aceptación o de integración al Agente Principal determinando la confianza del mismo.

Conclusiones:

De los diferentes experimentos realizados podemos determinar que el sistema de recomendación que utiliza Jaccard está en un nivel intermedio con respecto al resultado de los demás sistemas, pero conociendo el costo computacional de los sistemas que utilizan el algoritmo genético, consideramos que este presenta muy buenos resultados y eficiencia.

En el caso de utilizar Gemini se puede observar que los resultados con el algoritmo genético simple tienen resultados intermedios pero resulta mucho más costoso computacionalmente que el sistema ya mencionado. Para el algoritmo genético general se observa que el aumento de población normalmente mejora los resultados, al contrario del aumento de generaciones.

Por otro lado utilizando Mistral se observa disminución tanto al aumentar población, como generaciones, encontrándose resultados mejores que en el resto de sistemas utilizando este LLM, pero dado el costo computacional que supone utilizar LM Studio no recomendarías estos sistemas salvo que hubieran mostrado resultados mucho mejores o no exista otra opción de trabajo

Con respecto al análisis de datos de la simulación podemos concluir que el número de falsos negativos es bastante alto en los Agentes Difuso y de Comentarios, lo que esta afectando el resultado del Agente Principal, por otro lado el Agente Vectorial puede estar siendo demasiado permisivo con sus criterios de aceptación, consideramos que

sería productivo realizar experimentos donde se recomienden películas que sabemos que no le gustan a un determinado usuario para valorar la cantidad de falsos positivos de este agente.

Cambiando estos elementos de cada agente luego sería tarea evaluar el peso de cada uno en la decisión final del Agente Principal, en caso de que este siga presentando resultados similares.

Bibliografía:

1. Bag Sujoy, Kumar Krishna Sri, Tiwari Kumar Manoj (mayo 2019): An efficient recommendation generation using relevant Jaccard similarity. Url: https://www.sciencedirect.com/science/article/abs/pii/S0020025519300325
2. Silva B, Tsang Ren, Cavalcanti G, Tsang Jyh (2010): A graph-based friend recommendation system using Genetic Algorithm.
3. Gosh, S., Nahar, N., Wahab, M.A., Biswas, M., Hossain, M.S., & Andersson, K. (2020): Recommendation System for E-commerce Using Alternating Least Squares (ALS) on Apache Spark.
4. Google Developers: Get started with the Gemini API: Python
5. Nuñez Flores, Vergara Ortiz, Bocanegra Garcia (2014): Sistema experto basado en lógica difusa tipo 1 para determinar el grado de riesgo de preeclampsia. Url: https://revistascientificas.cuc.edu.co/index.php/ingecuc/article/view/341