ANÁLISIS DE REDES COMPLEJAS

Proyecto Final

Ernesto Corona Macías
Raúl Almeida López

La Habana
26 de junio de 2023

1 Generación de grafos.

1.1 Limpieza del set de datos.

Para este proyecto se utilizó un set de datos extraído de la plataforma para desarrollo de Data Science: Kaggle. Sin embargo, el set de datos obtenido aún requería ser tratado antes de poder ser utilizado adecuadamente para la construcción de los grafos. En este sentido se programaron tres funciones encargadas de obtener las recetas válidas, las reseñas correspondientes a dichas recetas, así como de extraer las reseñas que establecen relaciones de sustitución entre ingredientes.

Para enmarcar una receta como válida se utilizó como criterio, primero, la revisión de su listado de ingredientes descartando la receta en caso de contenido nulo. Luego, se establecieron los datos de interés para cada receta y se descartan nuevamente aquellas cuyos valores no estén definidos para estos parámetros. Finalmente, se reinicia el índice y se guarda en un archivo nuevo el set de datos con recetas listas para ser analizadas.

```python
def filter_recipes(df_recipes):
    mask = df_recipes["RecipeIngredientParts"].apply(lambda x: len(x) == 0)
    df_recipes = df_recipes.drop(df_recipes[mask].index)
    selected_columns = ["RecipeId","Name","TotalTime","Description","RecipeCategory","Keywords",
                        "RecipeIngredientParts","AggregatedRating","ReviewCount","Calories",
                        "FatContent","SaturatedFatContent","CholesterolContent",
                        "SodiumContent","CarbohydrateContent","FiberContent","SugarContent",
                        "ProteinContent","RecipeServings","RecipeInstructions"]
    df_recipes = df_recipes[selected_columns]
    df_recipes = df_recipes.dropna(axis=0)
    df_recipes = df_recipes.reset_index(drop=True)
    df_recipes.to_parquet("cleaned_recipes.parquet")
```

Figure 1: Función para limpiar el set de datos de recetas.

Para los datos de las reseñas simplemente se realiza una fusión de los datos originales con el nuevo set de recetas tomando como referencia de conexión el "RecipeId" de las recetas que se conservaron, descartando así todos las reseñas que no pertenezcan a dichas recetas. Luego, se reinicia el índice, se establecen los parámetros de interés para cada reseña y se guardan en un set de datos nuevo.

```python
def filter_reviews(df_recipes, df_reviews):
    df_reviews_filtrado = pd.merge(df_recipes[["RecipeId"]], df_reviews, on = "RecipeId", how = "inner")
    df_reviews_filtrado = df_reviews_filtrado.reset_index(drop = True)
    columns = ["RecipeId", "ReviewId", "AuthorId", "AuthorName", "Rating", "Review"]
    df_reviews_filtrado = df_reviews_filtrado[columns]
    df_reviews_filtrado.to_parquet("cleaned_reviews.parquet")
```

Figure 2: Función para limpiar el set de datos de reseñas.

Para el establecimiento en una reseña del carácter de sustitución de los ingredientes mencionados, se utilizó una expresión regular con palabras claves para filtrar todo el set de reseñas y almacenar las coincidencias con este criterio por separado.

```
def sustitution_reviews(df_reviews):
    df_reviews["Found"] = df_reviews["Review"].apply(lambda x: re.search(r"(.+\b(replace|change|substitute|swap)((\s+\b\w+\b){0,5})\s+(\bwith\b|\bfor\b|\binstead\b)((\s+\b\w+\b){0,5})\b)", x) != None)
    df_reviews = df_reviews.loc[df_reviews["Found"] == True]
    df_reviews.to_parquet("sustitutions.parquet")
```

Figure 3: Función para determinar reseñas que establezcan relaciones de sustitución.

1.2 Creación de los grafos.

Una vez han sido tratados los datos se puede proceder a la creación de los grafos que servirán de apoyo para el análisis. En este proyecto se definen cinco grafos fundamentales para extraer propiedades de los datos en cuestión.
El primer grafo mostrará la relación de pertenencia de cada receta con sus ingredientes.

```
def create_recipe_graph():
    df_recipes = pd.read_parquet("cleaned_recipes.parquet")
    G = nx.DiGraph()

    for n, recipe in df_recipes.iterrows():
        ingredients_set = get_ingredients(recipe)
        ingredients = list(ingredients_set)
        for i in range(len(ingredients)):
            G.add_edge(recipe["Name"], ingredients[i])

    nx.write_graphml(G, "Recipes.graphml")
```

Figure 4: Función para la creación del grafo receta-ingredientes.

El segundo grafo representa la relación entre ingredientes de una misma receta, para esto se utilizó como parámetro de peso entre conexiones el coeficiente pointwise mutual information(PMI). Este establece una medida estadística que mide la fuerza de la asociación entre dos términos, comparando su probabilidad conjunta con su probabilidad individual, la cuál puede ser obtenida del grafo receta-ingredientes ya creado.
El cálculo del PMI será una función auxiliar llamada dentro de la anterior y en conjunto ambas funciones quedarían definidas de la siguiente manera.

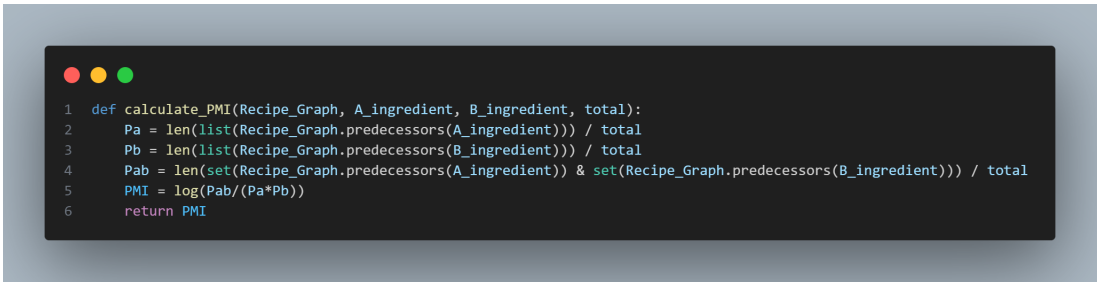
```python
def create_ingredients_graph():
    df_recipes = pd.read_parquet("cleaned_recipes.parquet")
    Recipe_Graph = nx.read_graphml("Recipes.graphml")
    G = nx.Graph()

    recipes_total = len(df_recipes.index)

    for _, recipe in df_recipes.iterrows():
        ingredients_set = get_ingredients(recipe)
        ingredients = list(ingredients_set)
        for i in range(len(ingredients)-1):
            for j in range(i+1,len(ingredients)):
                if G.has_edge(ingredients[i],ingredients[j]):
                    pass
                else:
                    PMI = calculate_PMI(Recipe_Graph, ingredients[i], ingredients[j], recipes_total)
                    G.add_edge(ingredients[i], ingredients[j], weight = PMI)

    nx.write_graphml(G, "Ingredients.graphml")
```

Figure 5: Función para la creación del grafo de relación de ingredientes.



```python
def calculate_PMI(Recipe_Graph, A_ingredient, B_ingredient, total):
    Pa = len(list(Recipe_Graph.predecessors(A_ingredient))) / total
    Pb = len(list(Recipe_Graph.predecessors(B_ingredient))) / total
    Pab = len(set(Recipe_Graph.predecessors(A_ingredient)) & set(Recipe_Graph.predecessors(B_ingredient))) / total
    PMI = log(Pab/(Pa*Pb))
    return PMI
```

Figure 6: Función para calcular PMI.

El tercer grafo, o grafo de revisores establecerá conexiones entre recetas que compartan muchas interacciones de usuarios comunes. De esta forma a partir del grafo obtenido podremos establecer preferencias de consumo de los usuarios y recomendar recetas partiendo de este criterio. Para lograrlo se estableció un umbral para recetas con al menos 10 reseñas, de las cuáles al menos 5 deben ser realizadas por usuarios comunes para conectarlas(Figura 7).

El cuarto grafo indica las relaciones entre ingredientes que se encuentran presentes en reseñas que ya han quedado establecidas con carácter de sustitución. Para ello en la función(Figura 8) primero se definen todos los ingredientes presentes en el grafo del mismo nombre, luego se itera por todas las reseñas de sustitución buscando coincidencias con los nombres de ingredientes, en caso de encontrar al menos dos en una misma reseña establece un vínculo entre dichos ingredientes con peso de valor 1, el cuál se verá incrementado de forma unitaria en caso de repetición posterior del par.

```python
def create_reviewers_graph():
    df_recipes_all = pd.read_parquet("cleaned_recipes.parquet")
    df_reviews = pd.read_parquet("cleaned_reviews.parquet")
    G = nx.Graph()

    df_recipes_all = df_recipes_all[df_recipes_all["ReviewCount"] > 10]
    df_recipes_all["Reviewers"] = df_recipes_all["RecipeId"].apply(lambda x: get_reviewers_id(x, df_reviews))
    print(df_recipes_all["Reviewers"])
    for n, recipe in df_recipes_all.iterrows():
        df_recipes = df_recipes_all.copy()
        df_recipes["AuthorsMatch"] = df_recipes["Reviewers"].apply(lambda x: len(x.intersection(recipe["Reviewers"])) > 5)
        df_recipes = df_recipes.loc[df_recipes["AuthorsMatch"] == True]
        for m, match in df_recipes.iterrows():
            if recipe["RecipeId"] != match["RecipeId"]:
                G.add_edge(recipe["Name"], match["Name"])
    nx.write_graphml(G, "Reviewers.graphml")
```

Figure 7: Función para la creación del grafo de recetas por interacción de usuarios.

```python
def create_sustitutions_graph():
    G = nx.read_graphml("Ingredients.graphml")
    sustitutions_graph = nx.Graph()
    df_sustitutions = pd.read_parquet("sustitutions.parquet")
    ingredients = []

    for node in G.nodes():
        ingredients.append(str(node))

    for _, review in df_sustitutions.iterrows():
        matches = []
        for i in ingredients:
            if i in review["Review"]:
                matches.append(i)
        if len(matches) > 1:
            for i in range(len(matches) - 1):
                for j in range(i+1, len(matches)):
                    if sustitutions_graph.has_edge(matches[i],matches[j]):
                        w = sustitutions_graph[matches[i]][matches[j]]['weight']
                        sustitutions_graph.add_edge(matches[i], matches[j], weight = w + 1)
                    else:
                        sustitutions_graph.add_edge(matches[i], matches[j], weight = 1)
    nx.write_graphml(sustitutions_graph, "Sustitutions.graphml")
```

Figure 8: Función para la creación del grafo de sustituciones de ingredientes.

Finalmente la función para la creación del último grafo(Figura 9), partirá del grafo de sustituciones realizando una limpieza de sus datos y detectando las posibles comunidades de ingredientes sustituibles que posea. Para ello, removerá del mismo todas las aristas cuyo peso sea inferior a 5, consolidando así que la relación de sustitución de ambos ingredientes fue dictaminada por varios usuarios. Posteriormente se eliminan todos los nodos que hayan quedado aislados y se procede a aplicar un algoritmo de detección de comunidades. En este caso se seleccionó el algoritmo de Louvain debido a su eficiencia y flexibilidad para adaptarse a todo tipo de grafos con diferentes tamaños y densidades. Este algoritmo tiende a producir particiones de alta calidad que maximizan la modularidad. Algunas desventajas de su uso son que no garantiza la solución óptima global, sino que encuentra una partición

que maximiza la modularidad en cada paso, lo que puede resultar en una solución subóptima en algunos casos, y la sensibilidad a la resolución, es decir, a la escala de la estructura del grafo que se está analizando, lo que puede afectar la calidad de la partición, tendiendo a descartar comunidades pequeñas según el volumen de información que maneje el grafo. Estas dos desventajas fueron valoradas y analizadas visualmente con la herramienta Gephi(Figura 10) pudiendo observarse valores de modularidad próximos a 0,3, además que se separan adecuadamente las comunidades pequeñas y las relaciones detectadas dentro de comunidades presentan sentido lógico(Figura 11).

```python
def clean_sustitutions_graph():
    G = nx.read_graphml("Sustitutions.graphml")

    edges_to_remove = [(u, v) for u, v, d in G.edges(data = True) if d['weight'] < 5]
    G.remove_edges_from(edges_to_remove)

    isolated_nodes = [n for n in G.nodes() if G.degree(n) == 0]
    G.remove_nodes_from(isolated_nodes)

    communities = list(louvain_communities(G))
    for i,comm in enumerate(communities):
        for node in G.nodes():
            if node in comm:
                nx.set_node_attributes(G, {node: {'color': i}})
    nx.write_graphml(G, "Sustitutions_communities.graphml")
```

Figure 9: Función para la creación del grafo de sustituciones con comunidades.

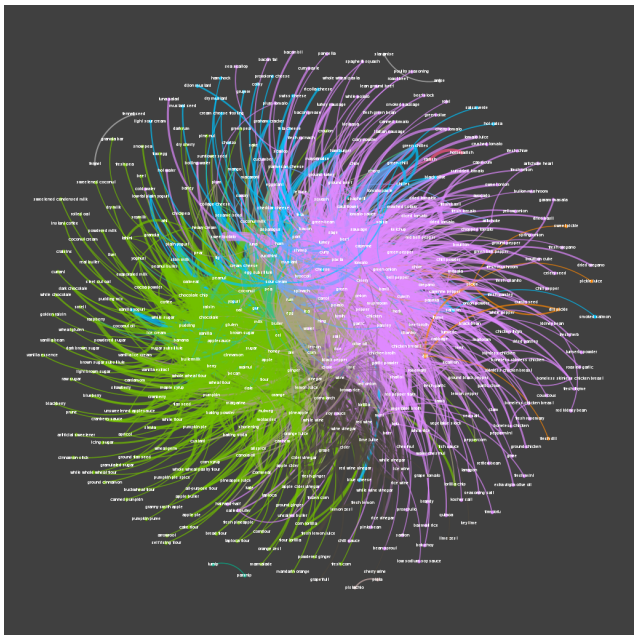


Figure 10: Grafo de sustituciones de ingredientes con comunidades detectadas.

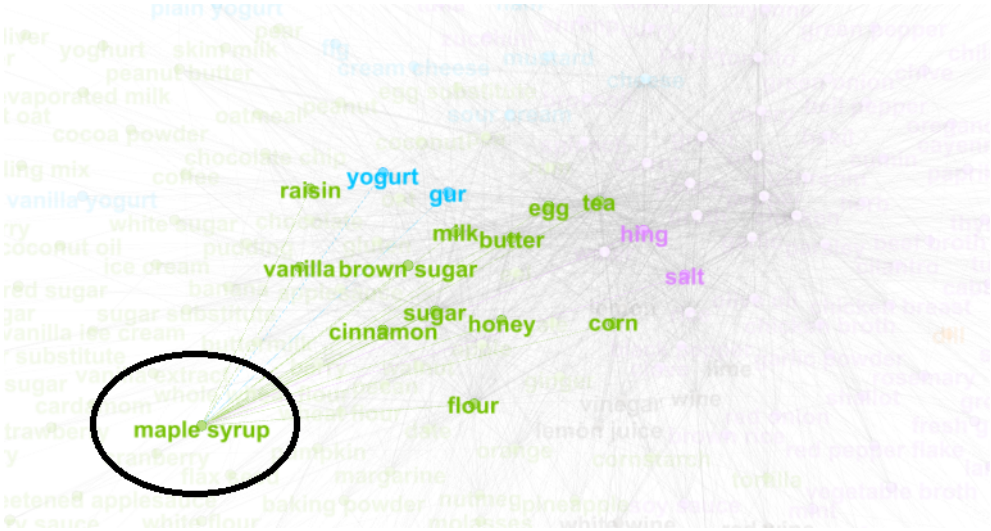


Figure 11: Relación de un ingrediente con sus vecinos. Apreciable similaridad dentro de elementos de una misma comunidad.

1.3 Funciones auxiliares.

Durante el proceso de construcción de algunos grafos es necesario extraer del set de recetas original, los ingredientes que pertenecen a cada una. Para ello se tuvo en cuenta que los posibles nombres de ingredientes podrían estar presentes en distintas variantes, por tanto, se hizo necesario realizar un proceso de lematización de las cadenas de texto correspondientes a sus nombres. Para lograr esta tarea, primero se dividen todas las cadenas de texto en tokens con la función interna de la biblioteca nltk, se procede entonces a eliminar todas las "Stop Words" que son básicamente conjunciones, artículos, preposiciones, adverbios, etc. Luego, para las palabras restantes se encuentran sus lemas y se vuelve a juntar la cadena obteniendo el nombre lematizado de cada ingrediente. Una vez obtenido se procede a retornar un set de ingredientes procesados. Las funciones a continuación se encargan de este proceso.

```python
def get_ingredients(recipe):
    ingredients = set()
    for ingredient in recipe["RecipeIngredientParts"]:
        ingredients.add(ingredient)
    ingredients = clean(ingredients)
    return ingredients
```

Figure 12: Función que devuelve un set de ingredientes procesados para cada receta.

```python
def clean(ingredients):
    tokenized = [nltk.word_tokenize(i.lower()) for i in ingredients]

    stop_words = set(stopwords.words("english"))
    stop_words.update(['.', ',', '"', "'", '?', '!', ':', ';', '(', ')', '[', ']', '{', '}'])
    non_stop_tokens = [[w for w in i if not w in stop_words] for i in tokenized]

    lemmatizer = WordNetLemmatizer()
    lemmatizeds = [[lemmatizer.lemmatize(w) for w in i] for i in non_stop_tokens]

    cleaned_ingredients = set()
    for trated_ingredients in lemmatizeds:
        trated_ingredients = " ".join(trated_ingredients)
        cleaned_ingredients.add(trated_ingredients)
    return cleaned_ingredients
```

Figure 13: Función que lematiza cada conjunto de ingredientes.

2 Análisis de similaridad de recetas.

La similaridad de cada receta con sus semejantes puede ser establecida de disimiles maneras. En este proyecto se calculó de 4 formas distintas teniendo en cuenta: coincidencias en recetas de vecinos de mayor PMI de cada ingrediente, contenido nutricional de la receta, interacciones de usuarios entre recetas y distancia normalizada entre ingredientes de recetas.

2.1 Análisis por PMI de ingredientes.

Para llevar a cabo este análisis se establece una función que extrae de la receta de interés todos sus ingredientes. Luego, a través del grafo de ingredientes obtiene para cada uno su vecino cuya conexión tenga mayor PMI y utiliza estos dos datos en conjunto para encontrar las recetas que mayores coincidencias de ingredientes otorguen, se ordena todo el set de recetas en base a este parámetro y se devuelve con un número n de posiciones(Figura 14).

2.2 Análisis por contenido nutricional de la receta.

Para este análisis lo primero que debe hacerse es a través de una función(Figura 15), limitar el set de recetas a los datos pertinentes a la nutrición. Luego se estandarizan estos datos dividiéndolos por la cantidad de porciones que permite la receta. Luego se pasan por un modelo de recomendación(Figura 16) que se encargará primero, de escalar los parámetros con el método MinMaxScaler y luego, ajustará el modelo con el método NearestNeighbors en base a la distancia euclideana. Se define entonces una función para este modelo que devuelva los k-vecinos más similares a una receta determinada. Esta vecindad se recibe y se fusiona con los datos originales a partir del parámetro "RecipeId", obteniendo nuevamente todas las propiedades de las recetas pero solo para la vecindad.

```python
def calculate_PMI_neighbors(Ingredients_Graph, df_recipes, recipe_id):
    PMI_ingredients = set()
    n = 4999

    recipe = df_recipes.loc[df_recipes['RecipeId'] == recipe_id].iloc[0]
    ingredients = get_ingredients(recipe)

    for i in ingredients:
        neighbors = Ingredients_Graph.neighbors(i)
        PMI_ingredients.add(i)

        max_PMI_neighbor = None
        max_weight = 0
        for neighbor in neighbors:
            weight = Ingredients_Graph[i][neighbor]['weight']
            if weight > max_weight:
                max_PMI_neighbor = neighbor
                max_weight = weight
                PMI_ingredients.add(max_PMI_neighbor)

    df_recipes["Coincidences"] = df_recipes["RecipeIngredientParts"].apply(lambda x: score_recipe_ingredients(x, PMI_ingredients))
    df_recipes_PMI = df_recipes.sort_values("Coincidences", ascending = False)
    df_recipes_PMI = df_recipes_PMI.iloc[0:n]
    return df_recipes_PMI, recipe_id
```

Figure 14: Función que encuentra la recetas con mayores coincidencias de ingredientes de mayor PMI.

```python
def calculate_nutritional_similarity(df_recipes, recipe_id):
    nutritional_columns = ["RecipeId", "Name", "Calories", "FatContent", "SaturatedFatContent", "CholesterolContent",
                           "SodiumContent", "CarbohydrateContent","FiberContent", "SugarContent", "ProteinContent",
                           "RecipeServings"]
    nutritional_df = df_recipes[nutritional_columns]
    nutritional_data_columns = ["Calories", "FatContent", "SaturatedFatContent", "CholesterolContent", "SodiumContent",
                      "CarbohydrateContent", "FiberContent", "SugarContent", "ProteinContent"]
    nutritional_df.loc[:,nutritional_data_columns] = nutritional_df.loc[:,nutritional_data_columns].div(nutritional_df["RecipeServings"], axis=0)
    nutritional_df.drop("RecipeServings", axis=1)
    recommender = NutritionalRecommender(nutritional_df, nutritional_data_columns)
    nutritional_sim_df = recommender.find_closest_recipes(recipe_id, k = 500)
    df_recipes = pd.merge(nutritional_sim_df[["RecipeId"]], df_recipes, on = "RecipeId", how = "inner")
    return df_recipes, recipe_id
```

Figure 15: Función que encuentra las recetas con mayor similaridad nutricional.

```python
class NutritionalRecommender:
    def __init__(self, nutritional_df, nutritional_data_columns):
        self.original_nutritional_df = nutritional_df
        self.nutritional_df = nutritional_df.copy()
        self.nutritional_data_columns = nutritional_data_columns

        self.scaler = MinMaxScaler()
        self.nutritional_df[self.nutritional_data_columns] = self.scaler.fit_transform(self.nutritional_df[self.nutritional_data_columns])

        self.knn = NearestNeighbors(metric='euclidean')
        self.knn.fit(self.nutritional_df[self.nutritional_data_columns])

    def find_closest_recipes(self, recipe_id, k):
        input_recipe = self.nutritional_df.loc[self.nutritional_df["RecipeId"] == recipe_id, self.nutritional_data_columns]
        distances, indices = self.knn.kneighbors(input_recipe, n_neighbors=k+1)

        closest_indices = indices[0][0:]
        return self.original_nutritional_df.iloc[closest_indices]
```

Figure 16: Modelo de recomendación basado en k-neighbors.

2.3 Análisis de recetas por interacción de usuarios.

Para determinar la similaridad simplemente se utiliza el grafo de revisores teniendo en cuenta que cada receta se encuentra conectada a todas con las que comparte por encima de cierto umbral de usuarios en común. Por tanto, simplemente se analizan y extraen los vecinos de las recetas y se buscan sus propiedades, para devolver finalmente un set con todos los datos.

```python
def calculate_reviewers_similarity(Reviewers_Graph, df_recipes, recipe_id):
    df = pd.DataFrame()

    recipe = df_recipes.loc[df_recipes['RecipeId'] == recipe_id].iloc[0]
    neighbors = Reviewers_Graph.neighbors(recipe["Name"])
    for neighbor in neighbors:
        fila = df_recipes.loc[df_recipes["Name"] == str(neighbor)].iloc[0]
        fila_df = fila.to_frame().T
        df = pd.concat([df, fila_df], ignore_index=True)
    return df, recipe_id
```

Figure 17: Función que extrae todos las recetas de la vecindad basado en interacciones de usuarios.

2.4 Análisis de recetas por distancia normalizada de ingredientes.

Para encontrar similaridad basado en este criterio, se obtienen los ingredientes para la receta de interés y para el resto, luego se calcula el factor de cercanía para cada receta, que no es más que usar una función para encontrar los caminos cortos entre todos los ingredientes del par, y hallar el inverso de su promedio, de esa forma las recetas con menor promedio de caminos cortos obtendrán un mayor valor de cercanía. Luego, se ordenan todas las recetas

basado en este factor y se devuelve un set de datos con la cantidad de posiciones que se necesite.

```python
def calculate_ingredient_similarity(Ingredients_Graph, df_recipes, recipe_id):
    recipe = df_recipes.loc[df_recipes['RecipeId'] == recipe_id].iloc[0]

    df_recipes["factor"] = df_recipes.apply(lambda x: shortest_path_factor(Ingredients_Graph, recipe, x), axis = 1)
    recipe_factor_max = df_recipes.sort_values("factor", ascending = False)
    return recipe_factor_max.iloc[1:10], recipe_id
```

Figure 18: Función que extrae las recetas de mayor similaridad por distancia.

```python
def shortest_path_factor(G, A_Recipe, B_Recipe) -> float:
    if A_Recipe["RecipeId"] != B_Recipe["RecipeId"]:
        A_ingredients = get_ingredients(A_Recipe)
        B_ingredients = get_ingredients(B_Recipe)
        shortest_paths =  []

        for i in A_ingredients:
            for j in B_ingredients:
                try:
                    shortest_paths.append(nx.shortest_path_length(G, source = i, target = j))
                except:
                    factor = -1
                    return factor
        factor = sum(shortest_paths) / len(shortest_paths)
        factor = 1 / factor
        return factor
    else:
        factor = -1
        return factor
```

Figure 19: Función que calcula la similaridad por distancia de dos recetas.

3 Interfaz gráfica del motor de búsqueda.

Esta ventana tendrá un cuadro de diálogo, el mismo recibirá una cadena de texto que podrá tener dos formatos en dependencia del checkbox presente en la página. Un formato será el nombre de la receta respetando su escritura, lo cuál dará acceso a una lista de autocompletado en la parte inferior de la ventana donde se podrá clickear para seleccionar más rápidamente la receta objetivo(Figura 20). El segundo formato será una lista de ingredientes separados por coma(Figura 21). En el primer caso la búsqueda nos llevará directamente a la ventana de información de la receta(Figura 22). En el segundo caso nos llevará a una ventana de coincidencias(Figura 23) con el criterio de igualdad al listado de ingredientes que se introduce, en la misma se podrá clickear alguna de las recetas obteniendo así la misma ventana de

información que en el primer caso. Finalmente, una vez obtenida la información de la receta, se podrán utilizar los checkboxes de la ventana información para encontrar recomendaciones de nuevas recetas basados en los filtros de similaridad que quedaron preestablecidos(Figura 24).

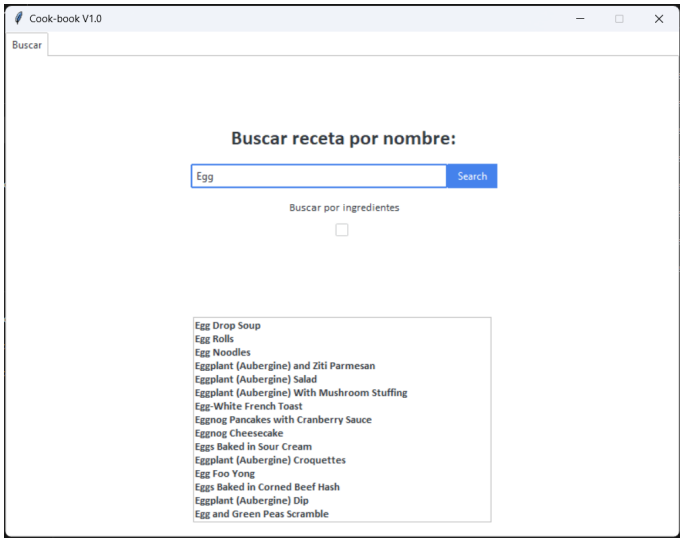


Figure 20: Búsqueda de receta por nombre.

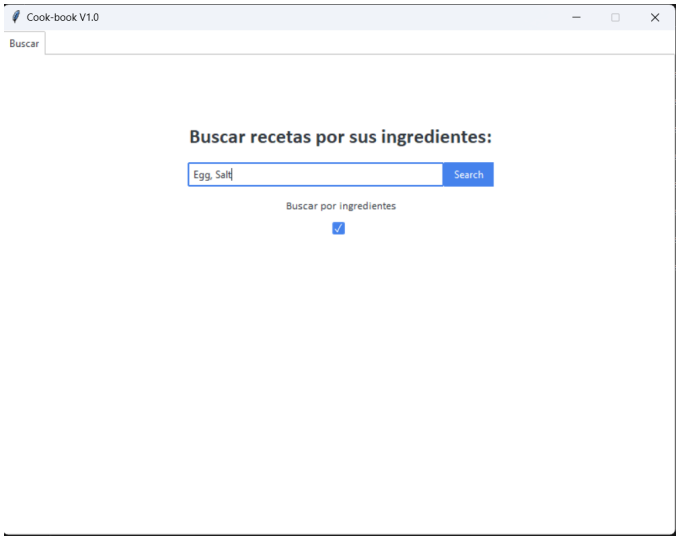


Figure 21: Búsqueda de receta por ingredientes.

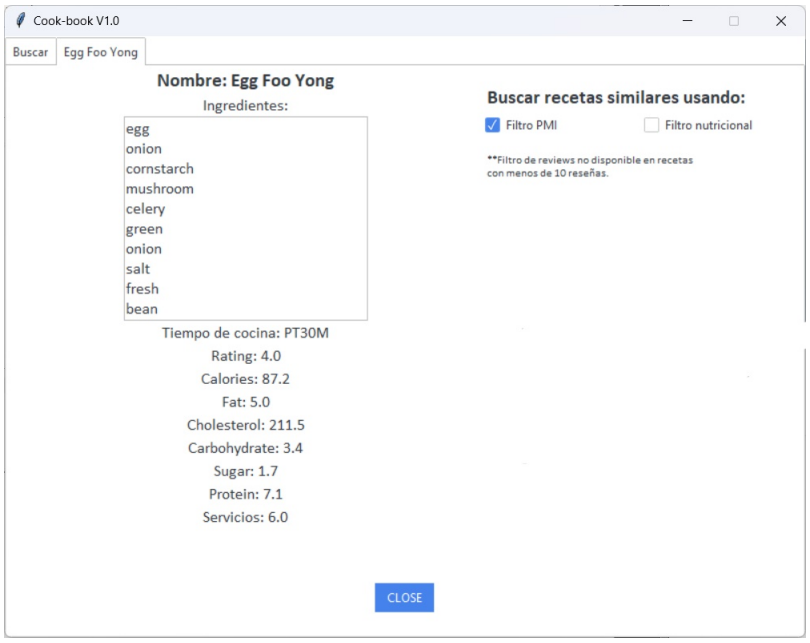


Figure 22: Información de la receta mostrada.

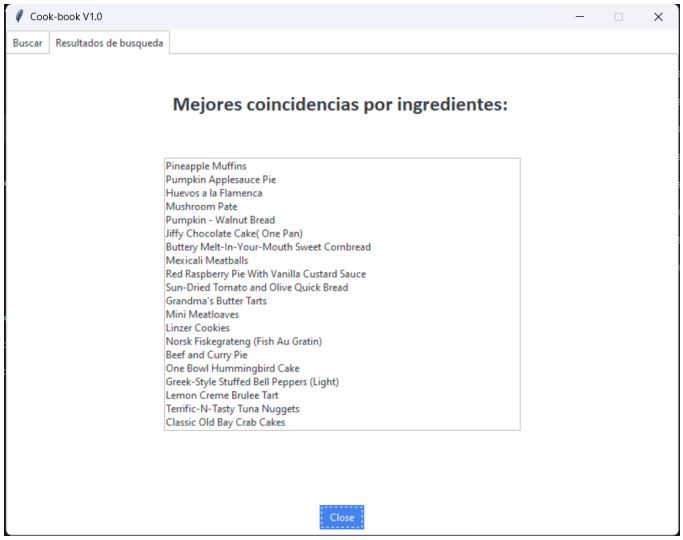


Figure 23: Listado de coincidencias de la búsqueda por ingredientes.

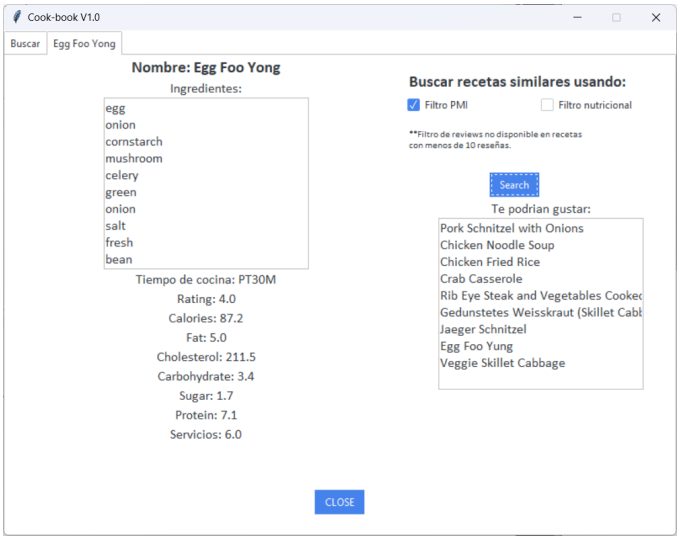


Figure 24: Recomendaciones otorgadas según filtro de búsqueda.