CourseMatch

¡Encuentra el mejor curso para ti!

Luca Santiago Ramirez Olivo
Sergio Gálvez Ortí

Universidad Europea de Madrid
Proyecto de Computación 1.

Índice

1 Introducción y descripción del problema abordado

Actualmente, encontramos una infinidad de posibilidades a la hora de formarnos sobre aquello que nos apasiona. Existen miles y miles de cursos de formación que abarcan prácticamente todas las áreas de estudio, lo que genera una gran duda acerca de cuál elegir. Dentro de la gran cantidad de opciones que existen, cada una de ellas nos ofrece unas características diferentes.
¿Cómo nos vamos a guiar para escoger la mejor opción para nosotros?

CourseMatch aborda este problema directamente, permitiendo al usuario encontrar los cursos que busca basándose en sus elecciones. CourseMatch se ajusta a la perfección a las características que el usuario necesita, filtrando su búsqueda entre los cursos que cumplen con ellas.

CourseMatch es un sistema de recomendación de cursos basado en técnicas de procesamiento de lenguaje natural (NLP), que se encarga de recibir qué tipo de cursos se buscan y ofrecer las mejores opciones basándose en unos parámetros básicos (Nombre del Curso, Habilidades Requeridas, Nivel de Dificultad y Plataforma)

El principal objetivo de CourseMatch es poder ofrecer y recomendar una amplia gama de cursos similares a los que el usuario busca, no una sola opción.

2 Metodología utilizada en la recopilación y procesamiento de los datos

En cuanto a la recopilación de los datos, el sistema de CourseMatch se basa principalmente en un dataset sobre cursos, que nos ofrece la siguiente información:

- Nombre del Curso
- Nivel de Dificultad
- Puntuación del Curso
- Habilidades
- Número de estudiantes
- Plataforma en la que se imparte

En cuanto al procesamiento de los datos, lo primero que hacemos es cargar el dataset y eliminar los duplicados con la siguiente función:

```
courses_path = 'data/courses_cleaned_dataset.csv'
courses_data = pd.read_csv(courses_path)
```

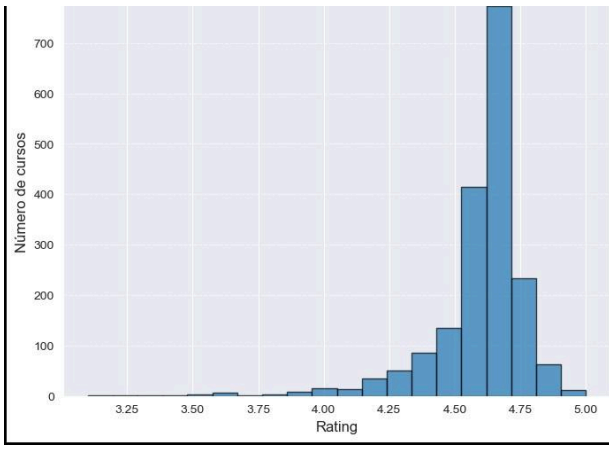
```
# Eliminar duplicados en el dataset base
courses_data = courses_data.drop_duplicates(subset='Course_Name').reset_index(drop=True)
```

Con esta función eliminamos del dataset aquella información que está duplicada y por tanto no nos vale.

2.1 Distribución de datos

Para poder visualizar los datos de una manera clara, hemos sacado las distribuciones de los datos del dataset, como se muestra en las imágenes:

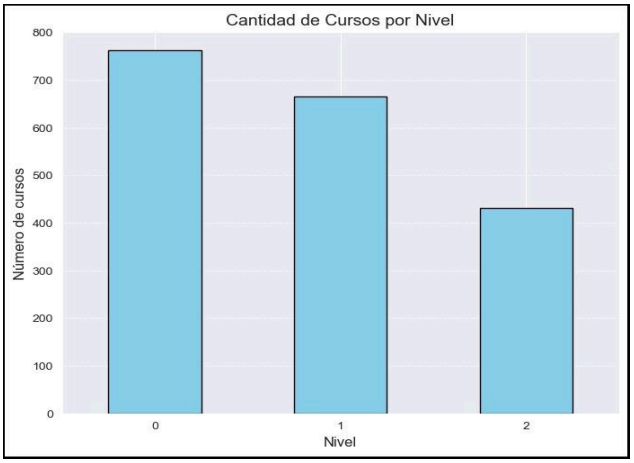
- **Para la columna de calificaciones:**



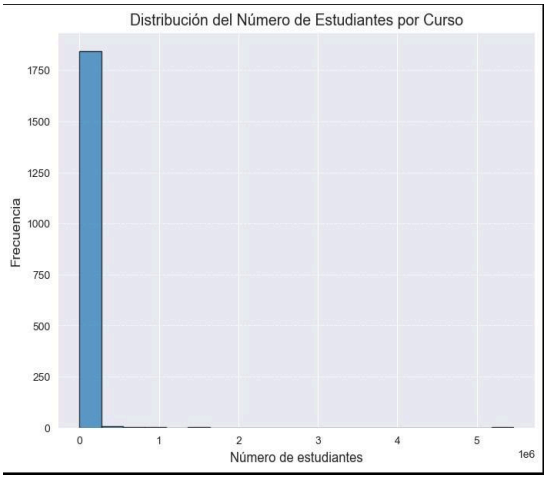

```
plt.figure(figsize=(8, 6))
plt.hist(courses_data['Rating'], bins=20, edgecolor='black', alpha=0.7)
plt.title('Distribución de Calificaciones', fontsize=14)
plt.xlabel('Rating', fontsize=12)
plt.ylabel('Número de cursos', fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

- **Para la columna de Nivel de dificultad** (Hemos contabilizado cantidad de cursos por nivel de dificultad)

```
plt.figure(figsize=(8, 6))
courses_data['Level'].value_counts().sort_index().plot(kind='bar', color='skyblue', edgecolor='black')
plt.title('Cantidad de Cursos por Nivel', fontsize=14)
plt.xlabel('Nivel', fontsize=12)
plt.ylabel('Número de cursos', fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.xticks(rotation=0)
plt.show()
```



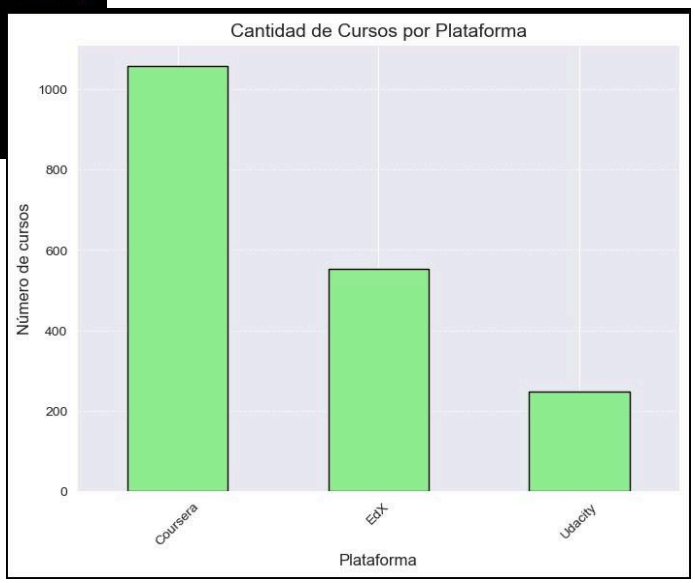
- **Para ver la popularidad:** (nº de estudiantes por curso)




```
plt.figure(figsize=(8, 6))
plt.hist(courses_data['Number of students'], bins=20, edgecolor='black', alpha=0.7)
plt.title('Distribución del Número de Estudiantes por Curso', fontsize=14)
plt.xlabel('Número de estudiantes', fontsize=12)
plt.ylabel('Frecuencia', fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

- **Para saber la cantidad de cursos por plataforma:**

```
plt.figure(figsize=(8, 6))
courses_data['Platform'].value_counts().plot(kind='bar', color='lightgreen', edgecolor='black')
plt.title('Cantidad de Cursos por Plataforma', fontsize=14)
plt.xlabel('Plataforma', fontsize=12)
plt.ylabel('Número de cursos', fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.xticks(rotation=45)
plt.show()
```



2.2 Procesamiento del texto en la columna Skills

Otra de las acciones que hay que hacer antes de pasar a trabajar con los datos, es fijarnos en la columna 'Skills' que se refiere a las habilidades desarrolladas en el curso, ya que es una columna que tiene texto y debemos trabajar con él y tiene que estar "limpio". Para ello, lo hemos hecho de la manera siguiente:

Limpieza del texto: Pasamos todo a minúsculas, eliminamos los caracteres especiales o aquellos que no tienen mucha importancia, y normalizamos el texto quitando los espacios extra.
Tras esto, buscaremos la forma de vectorizar estos textos. Utilizaremos la función TF-IDF que asigna valores numéricos a términos según su peso en cada documento.

```python
from sklearn.feature_extraction.text import TfidfVectorizer

# 1. Limpieza del texto
def clean_text(text):
    # Convertir a minúsculas y eliminar caracteres especiales
    return ' '.join(text.lower().replace(',', ' ').split())

# Aplicar limpieza al texto en la columna Skills
courses_data['Cleaned_Skills'] = courses_data['Skills'].apply(clean_text)

# 2. Vectorización con TF-IDF
tfidf_vectorizer = TfidfVectorizer(stop_words='english')  # Eliminar palabras comunes en inglés
tfidf_matrix = tfidf_vectorizer.fit_transform(courses_data['Cleaned_Skills'])

# Información sobre la matriz TF-IDF generada
tfidf_matrix.shape  # Dimensiones de la matriz
```

Con esto, hemos normalizado y limpiado las habilidades desarrolladas por cada curso representadas por la columna 'Skills'. La matriz TF-IDF tiene dimensiones (2377,3160), es decir tiene 2377 cursos que son todas las filas de la matriz, y tiene 3160 términos únicos que son las habilidades.
Entonces, cada curso está representado como un vector en un espacio de 3160 dimensiones (cada valor es el peso del término en el documento).

2.3 Estructuración y funcionamiento del programa

2.3.1 **Recommender** (recommender.py)

Una vez tenemos los datos limpios, pasamos a ver cómo funciona nuestro recomendador de cursos.
Principalmente, el programa funciona con su función principal, el recommender. Esta función es la que se encarga de:

```
# Validar entradas del usuario
if not keyword or not isinstance(keyword, str):
    raise ValueError("El parámetro 'keyword' debe ser una cadena no vacía.")
if level is not None and level not in courses_data['Level'].unique():
    raise ValueError(f"El parámetro 'level' debe ser uno de {courses_data['Level'].unique().tolist()} o None.")
if not isinstance(rating_range, tuple) or len(rating_range) != 2 or not (0.0 <= rating_range[0] <= 5.0) or not (0.0 <= rating_range[1] <= 5.0):
    raise ValueError("El parámetro 'rating_range' debe ser una tupla (min, max) con valores entre 0 y 5.")
if platform and platform not in courses_data['Platform'].unique():
    raise ValueError(f"El parámetro 'platform' debe ser una de las plataformas disponibles: {courses_data['Platform'].unique().tolist()} o None.")
if not isinstance(top_n, int) or top_n <= 0:
    raise ValueError("El parámetro 'top_n' debe ser un entero mayor a 0.")
```

Primero, comprueba si son válidas las entradas del usuario para poder proceder a la búsqueda

A continuación, filtra los cursos que se ajusten a los valores que se han indicado (ya validados)

```
# Filtro por palabra clave
filtered_courses = courses_data[
    courses_data['Cleaned_Skills'].str.contains(keyword.lower(), na=False)
]

# Filtro por nivel (si se especifica)
if level is not None:
    filtered_courses = filtered_courses[filtered_courses['Level'] == level]

# Filtro por rango de calificaciones
filtered_courses = filtered_courses[
    (filtered_courses['Rating'] >= rating_range[0]) & (filtered_courses['Rating'] <= rating_range[1])
]

# Filtro por plataforma (si se especifica)
if platform:
    filtered_courses = filtered_courses[filtered_courses['Platform'].str.contains(platform, na=False)]
```

Lanza un mensaje si no encuentra ninguno que cumpla con los datos dados.

```
# Si no hay cursos después del filtro, retornar un mensaje
if filtered_courses.empty:
    return "No se encontraron cursos que coincidan con los criterios especificados."
```

Después, lo siguiente es obtener el embedding del texto de la palabra clave que se busca. Esto nos permitirá más tarde obtener las similitudes y poder mostrar en pantalla los cursos más similares. Se calcula la similitud con la función similitud coseno. También se guardan las puntuaciones de similitud, para poder ofrecer esa información.

```
# Obtener embedding del keyword ingresado
keyword_embedding = model.encode(keyword.lower(), show_progress_bar=False)

# Calcular similitud con los cursos filtrados
filtered_embeddings = list(filtered_courses['Embeddings'])
similarity_scores = cosine_similarity([keyword_embedding], filtered_embeddings)[0]
```

Como se ve en el programa, en el sidebar de búsqueda, hay una barra para marcar cuánto peso tiene que tener la popularidad de los cursos en la recomendación. Para poder calcular esto, utilizamos la siguiente función.

```
# Incorporar popularidad en la ordenación
filtered_courses['Relevance'] = (
    (1 - popularity_weight) * filtered_courses['Similarity'] +
    popularity_weight * (filtered_courses['Number of students'] / filtered_courses['Number of students'].max())
)
```

Por último, el programa ordena las recomendaciones por Calificación y por Relevancia (calculada justo en el anterior bloque), ambas de manera descendente.

```
# Ordenar por relevancia y eliminar duplicados
recommendations = filtered_courses.sort_values(
    by=['Relevance', 'Rating'], ascending=[False, False]
).drop_duplicates(subset=['Course_Name', 'Platform']).head(top_n)

return recommendations[['Course_Name', 'Platform', 'Rating', 'Level', 'Skills', 'Similarity', 'Relevance']]
```

2.3.2 Preprocessor (preprocessor.py)

```
import os
import joblib
import pandas as pd
from sentence_transformers import SentenceTransformer

# Crear las carpetas necesarias
os.makedirs('data/processed', exist_ok=True)

# Cargar y procesar
def load_and_preprocess_data(file_path):
    courses_data = pd.read_csv(file_path)
    model = SentenceTransformer('all-MiniLM-L6-v2')

    # Limpieza y generación de embeddings
    def clean_text(text):
        return ' '.join(str(text).lower().replace(',', ' ').split())

    courses_data['Cleaned_Skills'] = courses_data['Skills'].apply(clean_text)
    courses_data['Embeddings'] = courses_data['Cleaned_Skills'].apply(
        lambda x: model.encode(x, show_progress_bar=False)
    )

    # Guardar el archivo
    joblib.dump(courses_data, 'data/processed/courses_data.pkl')
    return courses_data, model

# Ejecutar el preprocesamiento
load_and_preprocess_data('C:/Users/sergi/Desktop/ProyectoComp/CourseMatch-main/app/Model/data/courses_cleaned_dataset.csv')
```

Esta función nos ayuda a precargar los datos. Comienza cargando el dataset con la ruta especificada, y carga el modelo de lenguaje para utilizarlo en la fase de embeddings.
A continuación, limpia el texto de la columna skills y más tarde aplica la función ‘encode’ para guardarlo en los embeddings.

Finalmente guarda el archivo en un documento.

Por último, tenemos el programa principal, el cual gestiona una interfaz creativa que recoge los valores del usuario para las búsquedas, y después ofrece la información a los cursos que más se recomienden.

2.3.3 App (app/py)

Primero comienza cargando los datos preprocesados, y el modelo que se va a utilizar.

```python
# Cargar recursos
@st.cache_resource
def load_resources():
    courses_data = joblib.load('app/Model/data/processed/courses_data.pkl')  # Dataset preprocesado
    model = SentenceTransformer('all-MiniLM-L6-v2')  # Modelo de embeddings
    return courses_data, model

courses_data, model = load_resources()
```

Muestra un fragmento de HTML para dar la bienvenida. Además, muestra un resumen del dataset, con el total de cursos, las plataformas en las que se imparten,etc.

```python
# Mensaje de bienvenida
st.markdown(
    """
    <div style="background-color: #f4f4f4; padding: 15px; border-radius: 10px;">
        <h1 style="text-align: center; color: #4CAF50;">Bienvenido al Recomendador de Cursos</h1>
        <p style="text-align: center;">
            Esta aplicación utiliza inteligencia artificial para recomendarte cursos en línea
            basados en tus intereses, nivel de experiencia, y calificaciones de los usuarios.
        </p>
        <ul>
            <li>🔍 <b>Busca cursos</b> ingresando una palabra clave como <i>Python</i> o <i>JavaScript</i>.</li>
            <li>⚙️ <b>Personaliza</b> tus resultados ajustando filtros como nivel, plataforma y popularidad.</li>
            <li>📊 <b>Obtén recomendaciones</b> ordenadas por relevancia, popularidad y calificación.</li>
        </ul>
    </div>
    """,
    unsafe_allow_html=True
)
```

```python
# Resumen descriptivo del dataset
st.markdown("---")
st.markdown("### 📚 Resumen del Dataset")
total_courses = len(courses_data)
platform_counts = courses_data['Platform'].value_counts()
avg_rating = courses_data['Rating'].mean()
rating_range = (courses_data['Rating'].min(), courses_data['Rating'].max())
levels_available = courses_data['Level'].unique()

st.markdown(
    f"""
    - **Total de cursos disponibles:** {total_courses}
    - **Plataformas representadas:**
    """
)
for platform, count in platform_counts.items():
    st.markdown(f"  - {platform}: {count} cursos")
st.markdown(
    f"""
    - **Calificación promedio:** {avg_rating:.2f} (Rango: {rating_range[0]} - {rating_range[1]})
    - **Niveles disponibles:** {', '.join([f'Nivel {lvl}' for lvl in levels_available])}
    """
)
```

3 Detalles de los Modelos entrenados y el análisis de los datos obtenidos

Para comenzar el proyecto, desarrollamos un modelo de recomendación que funcionaba mediante la tabla TF-IDF. Este modelo trabaja con las frecuencias de aparición de los términos de búsqueda en las descripciones de los cursos o en los mismos nombres. Por ello, genera las recomendaciones en base a cuánto aparecen las palabras.
Al principio fue el mejor modelo que pensamos pero creíamos que podíamos mejorarlo un poco más, es decir, ver la posibilidad de que el sistema buscase algo más que frecuencias de palabras, ya que esto nos limita en cierta manera, cabiendo la posibilidad de tener bastantes casos en los que no encuentre ningún curso por varias razones (no se encuentre la palabra buscada específicamente, una errata en la búsqueda,etc.).

Tras esto, encontramos un modelo existente ya pre entrenado que funciona generando los embeddings de texto correspondientes a la búsqueda. Este modelo llamado 'all-MiniLM-L6-v2'. Este modelo nos permite encontrar similitudes semánticas entre los cursos y la búsqueda. Esto nos da una mejor cobertura a los posibles errores que veíamos.

Para poder tomar una decisión sobre qué modelo poder elegir, teníamos que probar la eficiencia de estos, y para ello sometimos al modelo a nuestro programa de pruebas:

3.1 Programa de Pruebas

- Modelos que evaluaremos
- Objetivos
- Métricas de Prueba
- Datos iniciales de Prueba
- Funciones Ejecutadas
- Resultados
- Conclusiones

3.1.1 Modelos que evaluaremos

Como hemos dicho anteriormente, durante este periodo de pruebas se evalúan dos modelos:

- Nuestro modelo entrenado de TF-IDF
- El modelo de embedding de texto ‘all-MiniLM-L6-v2’

3.1.2 Objetivos

Como principal objetivo de esta fase de pruebas y evaluación tenemos descubrir qué modelo funciona mejor y cuál nos ofrece unas recomendaciones más precisas. Necesitamos esto para poder tomar la decisión sobre qué modelo utilizar.

3.1.3 Métricas de Prueba

De entre todas las métricas posibles para medir nuestro modelo de recomendación, nosotros vamos a utilizar las siguientes:

- **NDCG (Normalized Discounted Cumulative Gain):** mide la calidad de las recomendaciones y se basa en la relevancia y en el orden de los resultados. (si salieran valores altos significaría que las recomendaciones más relevantes están mejor ordenadas)
- **Diversidad (Diversity):** Calcula cuánto de diferentes son las recomendaciones entre sí, es decir calcula la variedad de los resultados. (si salieran valores bajos sería que las recomendaciones son muy parecidas)

3.1.4 Datos iniciales de prueba

Para comenzar, tenemos que tener unos datos iniciales, que cuenten con una búsqueda y al menos algunas recomendaciones que nosotros creemos que son las más adecuadas (las llamaremos cursos esperados). Además contendrán las restricciones de Nivel, plataforma y rango de puntuación.
Por ejemplo: para ‘cybersec’ los cursos esperados son ‘Introduction to cybersecurity’ y ‘cybersecurity basics.’

```
test_cases = [
    {"keyword": "cybersec", "level": None, "rating_range": (4.0, 5.0), "platform": None, "relevant": {"Introduction to cybersecurity", "cybersecurity basics"}},
    {"keyword": "AI", "level": None, "rating_range": (3.0, 5.0), "platform": None, "relevant": {"Artificial Intelligence Essentials", "AI for Everyone"}},
    {"keyword": "data cleaning", "level": 1, "rating_range": (4.5, 5.0), "platform": "Coursera", "relevant": {"Data Visualization with Python", "Visualizing Data with R"}},
]
```

3.1.5 Funciones Ejecutadas

Para poder evaluar los modelos, primero calcularemos la diversidad con la siguiente función:

Función diversity_score:

Parámetros:
recommendations (list): Lista de los índices de los cursos recomendados
embeddings (list): Lista de embeddings de todos los cursos
k (int): Número de las recomendaciones que evaluaremos
Retorno:
float: Diversidad promedio

```python
def diversity_score(recommendations, embeddings, k=5):
    if len(recommendations) < 2:
        return 1.0  # Máxima diversidad si hay 1 o menos elementos

    selected_embeddings = [embeddings[idx] for idx in recommendations[:k]]
    similarities = cosine_similarity(selected_embeddings)
    avg_similarity = (similarities.sum() - similarities.trace()) / (len(selected_embeddings) * (len(selected_embeddings) - 1))
    return 1 - avg_similarity  # Diversidad es inversa de la similitud promedio
```

Esta función calcula la diversidad de los resultados basándose en las similitudes entre los cursos recomendados.
Primero comienza dando un resultado para cuando solo hay una o ninguna recomendación (<2), lo que significa la diversidad máxima, porque no hay otro resultado para comparar.

A continuación, se cogen los vectores de los embeddings de los cursos (solo los k primeros) y se calcula la similitud coseno entre los seleccionados (entre 0 y 1), se calcula la media de las similitudes (sin la diagonal, porque al ser los mismos es 1 siempre).
Por último, se calcula la diversidad utilizando la función: **diversidad = 1 - similitud**

Función evaluate_model:

Parámetros:

recommend_func(function): Función de recomendación del sistema principal
test_cases(list): lista de los datos iniciales de prueba
all_embeddings (list): Lista de todos los embeddings de todos los cursos
top_n (int): Número de recomendaciones que se van a evaluar

Retorno:

pd.DataFrame: Resultados de las métricas para cada caso de prueba

```python
from sklearn.metrics import ndcg_score

def evaluate_model(recommend_func, test_cases, all_embeddings, top_n=5):
    results = []
    for test in test_cases:
        keyword = test["keyword"]
        level = test.get("level", None)
        rating_range = test.get("rating_range", (0.0, 5.0))
        platform = test.get("platform", None)
        relevant = test["relevant"]

        # Obtener recomendaciones
        recommendations = recommend_func(
            keyword, level=level, rating_range=rating_range, platform=platform, top_n=top_n
        )

        if isinstance(recommendations, str):
            # Si no hay recomendaciones
            ndcg = 0
            diversity = 0
        else:
            recommended_courses = recommendations["Course_Name"].tolist()
            indices = recommendations.index.tolist()

            # Crear listas de relevancia binaria para NDCG
            relevance_binary = [1 if course in relevant else 0 for course in recommended_courses]
            relevance_binary += [0] * (top_n - len(relevance_binary))  # Asegurar tamaño top_n
            ground_truth = [1] * min(len(relevant), top_n) + [0] * max(0, top_n - len(relevant))
            ndcg = ndcg_score([ground_truth], [relevance_binary])
            diversity = diversity_score(indices, all_embeddings, k=top_n)

        results.append({
            "Keyword": keyword,
            "NDCG@5": ndcg,
            "Diversity": diversity,
        })

    return pd.DataFrame(results)
```

Como podemos observar, comienza iterando cada uno de los casos de ejemplo y guarda cada uno de los datos en sus variables. Después, calcula las recomendaciones (si no hay, diversidad y NDCG@5 son 0). Guarda los nombres de los cursos recomendados y sus índices que representan la posición.

Se guarda en un vector para indicar si están en la lista de cursos esperados o no (relevance_binary); y un vector binario para representar la relevancia esperada. Se utiliza la función ndcg_score calcula la métrica NDCG@5 con ambos vectores binarios. Se utiliza por último la función diversity para la diversidad y se guardan los resultados para posteriormente mostrarlos.

3.1.6 Resultados

Tras haber visto las dos funciones con las que evaluaremos ambos modelos en nuestro programa, las ponemos en función para ver los resultados:

```
tfidf_results = evaluate_model(recommend_courses, test_cases, all_embeddings, top_n=5)
embeddings_results = evaluate_model(recommend_courses_with_embeddings, test_cases, all_embeddings, top_n=5)

print("Resultados TF-IDF:")
print(tfidf_results)

print("\nResultados Embeddings:")
print(embeddings_results)
```

```
Resultados TF-IDF:
         Keyword    NDCG@5  Diversity
0       cybersec  0.723136   0.898148
1             AI  0.723136   0.898148
2  data cleaning  0.723136   0.898148

Resultados Embeddings:
         Keyword    NDCG@5  Diversity
0       cybersec  0.723136   0.508070
1             AI  0.723136   0.705304
2  data cleaning  0.723136   0.660656
```

Análisis de los resultados

Como podemos ver, para NDCG@5 ambos modelos tienen la misma puntuación **(0.723136)**, lo que nos dice que la calidad de las recomendaciones para ambos modelos es similar. Por otra parte, el modelo TF-IDF tiene una puntuación más alta en diversidad que el modelo de embeddings. Con esto afirmamos que las recomendaciones del modelo TF-IDF son más variadas y que el modelo de embeddings muestra recomendaciones más similares entre sí.

Sin embargo, al analizar estos resultados, pensamos que la diversidad entre recomendaciones no es nuestro principal objetivo, buscamos ofrecer una amplia cantidad de recomendaciones, pero precisas y semánticamente relevantes. Esto nos parece esencial a la hora de poder manejar consultas complejas o imprecisas, con las que el modelo de embeddings demuestra que es el más óptimo a la hora de captar relaciones implícitas entre conceptos.

3.1.7 Conclusiones

Tras haber comparado ambos modelos mediante dos de las métricas estándar para evaluar sistemas de recomendación (NDCG@5 y Diversity), llegamos a la conclusión de que el modelo de embeddings de texto es el que mejor se adapta a las necesidades de nuestro recomendador. Nos permite una gran capacidad para realizar búsquedas semánticas, garantizando resultados relevantes ante consultas que no se adaptan del todo con el contenido del dataset.

3.2 Ajustes tras Presentación de Prototipos

Tras la presentación de prototipos, estuvimos pensando la manera de poder mejorar nuestro sistema de recomendación, y nos dimos cuenta de la limitación en ciertos aspectos que sufrimos a la hora de la información que nos ofrece el dataset (siempre podría ser más). Pensamos muchas mejoras, así como que podríamos tratar de poner los links a los cursos que nuestro recomendador ofrece,dando la facilidad de acudir a él directamente. También pensamos en filtrar los cursos en un rango de precio del curso.
Todas estas mejoras nos resultan muy complicadas ya que es una información que no tenemos (intentamos agregar APi's de las plataformas de los cursos pero solo existe una y es de pago) y nos llevaría mucho tiempo recopilar.
La alternativa que hemos encontrado para mejorar nuestro sistema es mostrar las tendencias de los lenguajes de programación, así como cuál está en auge o cuál está en declive, cuál ha sido el más utilizado,etc.
Todos estos datos los hemos recogido de datos de encuestas de stack overflow de los años 2022, 2023 y 2024.

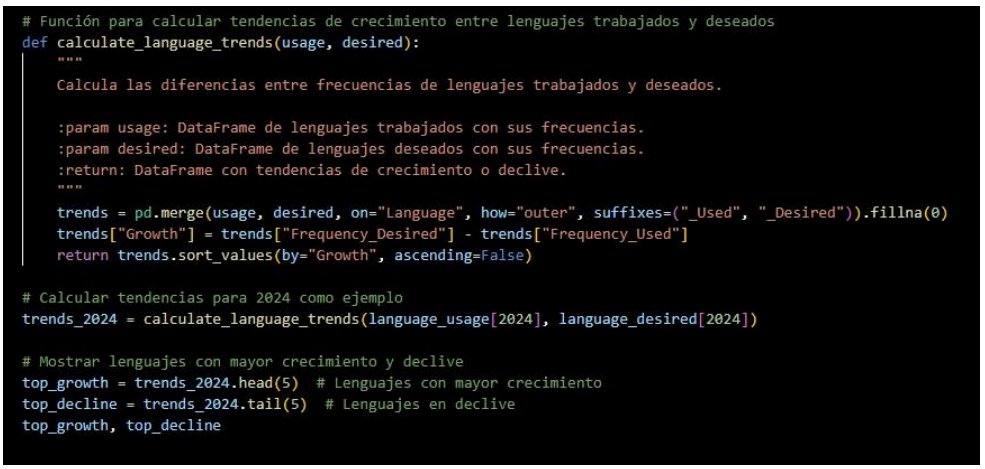
Lo primero que observamos en los datasets es que comparten algunas columnas clave, así como:

- LanguageHaveWorkedWith: Son los lenguajes trabajados actualmente (217.860 registros no nulos)
- LanguageWantToWorkWith: Lenguajes deseados (203.488 registros no nulos)
- DevType: Roles de desarrolladores (197.619 registros no nulos)
- LearnCode y LearnCodeOnline: Métodos de aprendizaje
- Year: Año correspondiente (esto se trabaja más para la tendencia)

Para poder integrar toda esta información en nuestra app, hemos añadido una nueva sección a la página principal, la cual presenta inicialmente dos modos de visualización de los datos: Gráficos o Rankings.

Hemos añadido toda la información en 3 desplegables:

- Lenguajes en Auge y Declive:
  Este desplegable muestra el crecimiento relativo de cada lenguaje (deseado - utilizado), para poder crear el Gráfico o Ranking de los lenguajes en auge o declive.



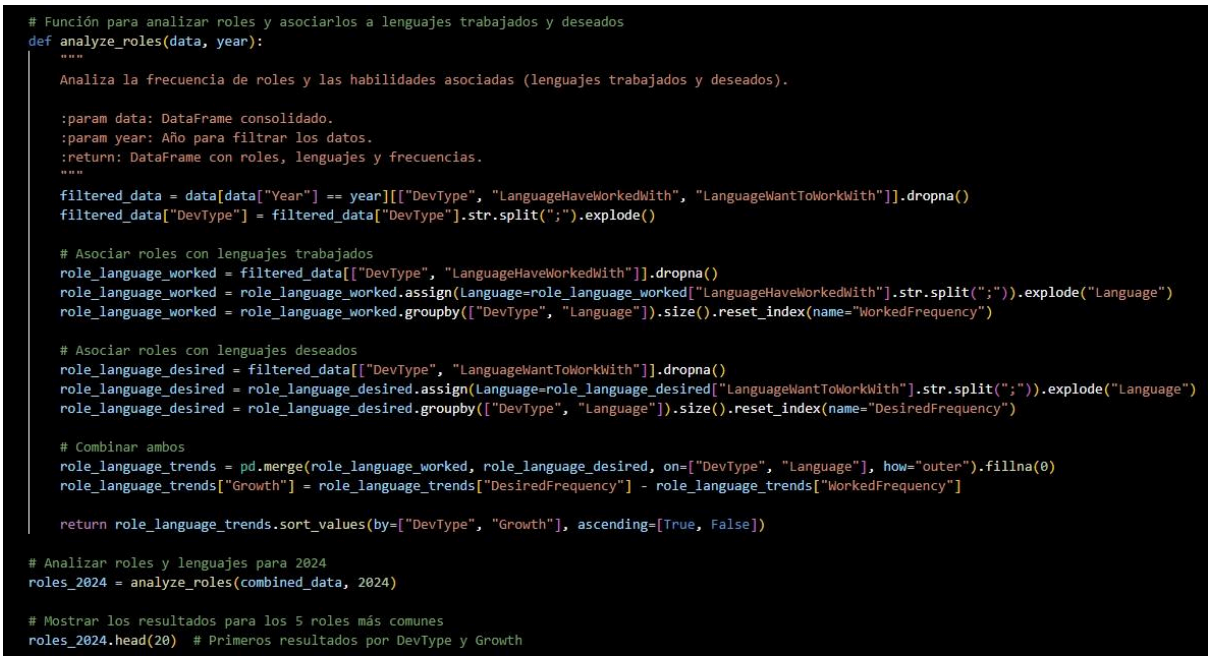
```python
# Función para calcular tendencias de crecimiento entre lenguajes trabajados y deseados
def calculate_language_trends(usage, desired):
    """
    Calcula las diferencias entre frecuencias de lenguajes trabajados y deseados.

    :param usage: DataFrame de lenguajes trabajados con sus frecuencias.
    :param desired: DataFrame de lenguajes deseados con sus frecuencias.
    :return: DataFrame con tendencias de crecimiento o declive.
    """
    trends = pd.merge(usage, desired, on="Language", how="outer", suffixes=("_Used", "_Desired")).fillna(0)
    trends["Growth"] = trends["Frequency_Desired"] - trends["Frequency_Used"]
    return trends.sort_values(by="Growth", ascending=False)

# Calcular tendencias para 2024 como ejemplo
trends_2024 = calculate_language_trends(language_usage[2024], language_desired[2024])

# Mostrar lenguajes con mayor crecimiento y declive
top_growth = trends_2024.head(5)  # Lenguajes con mayor crecimiento
top_decline = trends_2024.tail(5)  # Lenguajes en declive
top_growth, top_decline
```

- Relación entre Roles y Lenguajes Clave: En este desplegable, trabajamos la relación entre los roles de desarrolladores y los lenguajes utilizados y deseados. Con esto, identificamos los roles emergentes y sus habilidades clave.



```python
# Función para analizar roles y asociarlos a lenguajes trabajados y deseados
def analyze_roles(data, year):
    """
    Analiza la frecuencia de roles y las habilidades asociadas (lenguajes trabajados y deseados).

    :param data: DataFrame consolidado.
    :param year: Año para filtrar los datos.
    :return: DataFrame con roles, lenguajes y frecuencias.
    """
    filtered_data = data[data["Year"] == year][["DevType", "LanguageHaveWorkedWith", "LanguageWantToWorkWith"]].dropna()
    filtered_data["DevType"] = filtered_data["DevType"].str.split(";").explode()

    # Asociar roles con lenguajes trabajados
    role_language_worked = filtered_data[["DevType", "LanguageHaveWorkedWith"]].dropna()
    role_language_worked = role_language_worked.assign(Language=role_language_worked["LanguageHaveWorkedWith"].str.split(";")).explode("Language")
    role_language_worked = role_language_worked.groupby(["DevType", "Language"]).size().reset_index(name="WorkedFrequency")

    # Asociar roles con lenguajes deseados
    role_language_desired = filtered_data[["DevType", "LanguageWantToWorkWith"]].dropna()
    role_language_desired = role_language_desired.assign(Language=role_language_desired["LanguageWantToWorkWith"].str.split(";")).explode("Language")
    role_language_desired = role_language_desired.groupby(["DevType", "Language"]).size().reset_index(name="DesiredFrequency")

    # Combinar ambos
    role_language_trends = pd.merge(role_language_worked, role_language_desired, on=["DevType", "Language"], how="outer").fillna(0)
    role_language_trends["Growth"] = role_language_trends["DesiredFrequency"] - role_language_trends["WorkedFrequency"]

    return role_language_trends.sort_values(by=["DevType", "Growth"], ascending=[True, False])

# Analizar roles y lenguajes para 2024
roles_2024 = analyze_roles(combined_data, 2024)

# Mostrar los resultados para los 5 roles más comunes
roles_2024.head(20)  # Primeros resultados por DevType y Growth
```

- Métodos de Aprendizaje más Populares: para mostrar esta información, hemos usado las columnas LearnCode y LearnCodeOnline. Lo hemos añadido a tendencias para poder informar sobre cómo prefieren aprender los usuarios.

```python
# Función para analizar métodos de aprendizaje preferidos
def analyze_learning_methods(data, year):
    """
    Analiza las preferencias de aprendizaje (métodos offline y online) para un año específico.

    :param data: DataFrame consolidado.
    :param year: Año para filtrar los datos.
    :return: DataFrame con métodos de aprendizaje y sus frecuencias.
    """
    filtered_data = data[data["Year"] == year]

    # Métodos offline (LearnCode)
    offline_methods = filtered_data["LearnCode"].dropna().str.split(";").explode()
    offline_frequencies = offline_methods.value_counts().reset_index()
    offline_frequencies.columns = ["Method", "OfflineFrequency"]

    # Métodos online (LearnCodeOnline)
    online_methods = filtered_data["LearnCodeOnline"].dropna().str.split(";").explode()
    online_frequencies = online_methods.value_counts().reset_index()
    online_frequencies.columns = ["Method", "OnlineFrequency"]

    # Combinar ambos
    learning_methods = pd.merge(offline_frequencies, online_frequencies, on="Method", how="outer").fillna(0)
    learning_methods["TotalFrequency"] = learning_methods["OfflineFrequency"] + learning_methods["OnlineFrequency"]

    return learning_methods.sort_values(by="TotalFrequency", ascending=False)

# Analizar métodos de aprendizaje para 2024
learning_methods_2024 = analyze_learning_methods(combined_data, 2024)

# Mostrar los métodos más populares
learning_methods_2024.head(10)
```

4. Instrucciones para desplegar el proyecto

En cuanto al manual del desarrollador está realizado en una presentación en pdf de una manera más visual e intuitiva, para facilitar al desarrollador y hacerlo más sencillo.
Todos los pasos se realizan desde el cmd.
Si lo has descargado desde el zip, empezar en el paso 3 (entorno virtual).

Comandos cmd:

1º: *cd **'ruta a carpeta de destino del proyecto'***
2º: *git clone https://github.com/lucaramirezo/CourseMatch.git*
3º: *cd CourseMatch*
4º: *python -m venv env*
5º: *.\\env\\Scripts||activate*
6º: *pip install -r requirements.txt*
7º: *python setup_surveys.py*
8º: *python app/Model/preprocessor.py*
9º: *streamlit run app/app.py*

5. Instrucciones de uso del prototipo y conclusiones

Para el manual del usuario, lo tenemos también en un pdf en el que se observa cómo funciona el proyecto de una manera muy intuitiva y fácil de entender.
Ambos documentos están dentro del .zip del proyecto entregado.

Gracias a este proyecto, hemos podido aprender acerca de los diferentes tipos de modelo de recomendación. Sin embargo, lo más importante que hemos aprendido es la vitalidad de analizar detalladamente las necesidades de cada proyecto, es decir, saber qué necesitamos para poder ofrecer la solución más óptima. Esto lo hemos visto en nuestro proyecto a la hora de elegir entre los dos modelos con los que contábamos. Poder evaluar y pensar qué es realmente lo que queremos y cómo queremos resolverlo, nos ayudó bastante a decidirnos sobre cuál era la mejor opción de modelo a aplicar en nuestro sistema de recomendación.

6. Bibliografía

Modelo TF-IDF:
Guzmán Luna, R., & Pazos Rangel, R. A. (2013). Aplicación del factor TF-IDF en el análisis semántico de una colección documental. *Redalyc. Revista de Ciencias de la Información*, 4(2), 1–10. Recuperado de https://www.redalyc.org/pdf/161/16143063001.pdf
Modelo Embeddings:
Hugging Face. (n.d.). Sentence Transformers: all-MiniLM-L6-v2. Encontrado en https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2
Dataset Cursos:
Nomanturag. (n.d.). Online course dataset: edX, Udacity, Coursera. Descargado desde https://www.kaggle.com/datasets/nomanturag/online-course-datasetedx-udacity-coursera
Link Datos de Encuestas Stack Overflow:
Stack Overflow. (n.d.). Stack Overflow Developer Survey. Encuestas descargadas en https://survey.stackoverflow.co/
Python:
Python Software Foundation.*Python programming language*. Encontrado en https://www.python.org/
Streamlit:
Streamlit Inc. (n.d.). *Streamlit: The fastest way to build and share data apps*. Descargado de https://streamlit.io/
Pandas:
The pandas development team. (s.f.). pandas: Python Data Analysis Library. Encontrado en https://pandas.pydata.org/docs/
Scikit-learn:
Scikit-learn developers. (s.f.). Scikit-learn: Machine Learning in Python. https://scikit-learn.org/stable/