UNIVERSIDAD DE LOS ANDES

NATURAL LANGUAGE PROCESSING

ISIS 4221

Tarea 1

Autores:
Isabella MARTÍNEZ
Gustavo MENDEZ
Paula DAZA
Nicolás KLOPSTOCK

Septiembre 4, 2024

Tabla de contenidos

Universidad de los Andes

1 Introducción

En esta tarea, se abordan diferentes aspectos relacionados con el procesamiento de lenguaje natural, incluyendo la implementación de métricas de evaluación de información, la construcción de índices invertidos y la recuperación de documentos utilizando diversas técnicas. El objetivo es aplicar estos conceptos para resolver problemas específicos en el ámbito de la recuperación de información (IR). Se exploran dos enfoques distintos para la recuperación de información: el modelo basado en la Representación Vectorial Documental Relevante (RRDV) y un modelo basado en la biblioteca Gensim, ampliamente utilizada en el procesamiento de lenguaje natural. Ambos modelos emplean el cálculo de TF-IDF (Term Frequency-Inverse Document Frequency) para representar los documentos en un espacio vectorial y permiten realizar búsquedas eficientes mediante la similitud coseno. A lo largo del documento, se describen las implementaciones de los modelos, se presentan los resultados de las evaluaciones y se analizan las diferencias en el rendimiento de ambos enfoques. Finalmente, se discuten las implicaciones de estos resultados y se ofrecen conclusiones sobre la efectividad de cada modelo en el contexto de la recuperación de información.

2 Métricas de Evaluación

Las métricas de evaluación son esenciales para medir la efectividad de un sistema de recuperación de información. A continuación, se describen las principales métricas utilizadas en este trabajo, junto con sus definiciones matemáticas e implementaciones en Python.

2.1 Precisión

La **Precisión** se define como la proporción de documentos recuperados que son relevantes.

$$\mathcal{P} = \frac{|\text{RET} \cap \text{REL}|}{|\text{RET}|} \tag{1}$$

Donde $|\text{RET}|$ representa el número de documentos recuperados, y $|\text{RET} \cap \text{REL}|$ es el número de documentos relevantes recuperados.

La implementación en Python se muestra a continuación:

```python
def precision(relevance_query: List[int]):
    assert set(relevance_query).issubset((0, 1)), "Only binary values (0, 1) allowed."

    return sum(relevance_query) / len(relevance_query)
```

2.2 Precisión en K

La **Precisión en K** se define como la proporción de los K documentos recuperados más relevantes.

```python
def precision_at_k(relevance_query: List[int], k: int):
    assert set(relevance_query).issubset((0, 1)), "Only binary values (0, 1) allowed."
    assert k > 0, "K must be greater or equal than 1."

    return sum(relevance_query[:k]) / len(relevance_query[:k])
```

2.3 Recall en K

El **Recall en K** se define como la proporción de documentos relevantes que se recuperan en los K primeros resultados.

```python
def recall_at_k(relevance_query: List[int], k: int, num_relevant_docs: int):
    assert set(relevance_query).issubset((0, 1)), "Only binary values (0, 1) allowed."
    assert k > 0, "K must be greater or equal than 1."
    assert num_relevant_docs > 0, "Number of relevant docs must be greater or equal than 1."

    return sum(relevance_query[:k]) / num_relevant_docs
```

2.4 Average Precision

La **Precisión Promedio** (*Average Precision*) se define como el promedio de las precisiones calculadas cada vez que se encuentra un documento relevante.

```python
def average_precision(relevance_query: List[int]):
    assert set(relevance_query).issubset((0, 1)), "Only binary values (0, 1) allowed."

    cumulative_precision = 0
    relevant_count = 0

    for observation_count, relevance in enumerate(relevance_query, 1):
        if relevance:
            relevant_count += 1
            cumulative_precision += relevant_count / observation_count

    return cumulative_precision / relevant_count
```

2.5 Mean Average Precision (MAP)

El **Mean Average Precision** (*MAP*) es el promedio de la Precisión Promedio calculado para varias consultas.

```python
def mean_average_precision(relevance_queries: List[List[int]]):
    assert all(set(sublist).issubset((0, 1)) for sublist in relevance_queries), "Only binary
    ↪  values (0, 1) allowed in all query results."

    return sum(average_precision(relevance_query) for relevance_query in relevance_queries) /
    ↪  len(relevance_queries)
```

2.6 Discounted Cumulative Gain at K (DCG@K)

El **Discounted Cumulative Gain at K** (*DCG@K*) mide la calidad de los documentos recuperados basado en la relevancia, ponderada por la posición del documento en la lista de resultados.

$$\text{DCG@K} = \sum_{i=1}^{K} \frac{\text{REL}_i}{\log_2(\max(i, 2))} \tag{2}$$

```python
def discounted_cumulative_gain(relevance_query: List[int], k: int):
    assert all(x >= 0 for x in relevance_query), "All elements must be integers greater than
    ↪  or equal to 0."
    assert k > 0, "K must be greater or equal than 1."

    return sum(relevance / np.log2(max(i, 2)) for i, relevance in
    ↪  enumerate(relevance_query[:k], 1))
```

2.7 Normalized Discounted Cumulative Gain at K (nDCG@K)

El **Normalized Discounted Cumulative Gain at K** (*nDCG@K*) normaliza el valor de DCG@K dividiéndolo por el mejor DCG@K posible para la consulta.

```python
def normalized_discounted_cumulative_gain(relevance_query: List[int], k: int):
    assert all(x >= 0 for x in relevance_query), "All elements must be integers greater than
    ↪  or equal to 0."
    assert k > 0, "K must be greater or equal than 1."

    rq = relevance_query.copy()
    rq.sort(reverse=True)
    return discounted_cumulative_gain(relevance_query, k) / discounted_cumulative_gain(rq, k)
```

3 Preparación de Texto

En esta sección se detalla el proceso de preparación de texto que se llevó a cabo antes de la implementación de los modelos de recuperación de información. Este proceso es crucial para asegurar que los textos sean adecuados para su análisis y recuperación.

3.1 Preprocesamiento de Tokens

Para el preprocesamiento de tokens se utilizó la librería `nltk`. Este proceso incluye las siguientes etapas:

1. **Conversión a minúsculas:** Todos los tokens se convierten a minúsculas para evitar diferencias causadas por la capitalización.
2. **Eliminación de stopwords:** Se eliminan las stopwords, que son palabras comunes que no aportan valor semántico significativo, como artículos y preposiciones.
3. **Eliminación de caracteres no alfabéticos:** Se eliminan todos los caracteres que no son letras, tales como puntuación y números.
4. **Aplicación de stemming:** Se aplica el algoritmo de *PorterStemmer* para reducir las palabras a su raíz morfológica, facilitando así la agrupación de términos similares.

El código utilizado para este preprocesamiento es el siguiente:

```python
def token_preprocessing(tokens: List[str]) -> List[str]:
    tokens = [word.lower() for word in tokens]
```

```
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]
    tokens = [re.sub(r'\W', '', word) for word in tokens]
    tokens = [re.sub(r'\s+[a-zA-Z]\s+', '', word) for word in tokens]
    tokens = [re.sub(r'\^[a-zA-Z]\s+', '', word) for word in tokens]
    tokens = [re.sub(r'[0-9]+', '', word) for word in tokens]
    tokens = [word for word in tokens if len(word) > 0]
    ps = PorterStemmer()
    return [ps.stem(word) for word in tokens]
```

3.2 Preprocesamiento de Texto

El proceso de preprocesamiento de texto se realiza utilizando la función `text_preprocessing`, que tokeniza el texto y aplica el preprocesamiento de tokens descrito anteriormente:

```
def text_preprocessing(text: str) -> List[str]:
    tokens = word_tokenize(text)
    return token_preprocessing(tokens)
```

3.3 Clase `Document`

Se definió una clase `Document` para almacenar y procesar los documentos. Cada objeto de esta clase almacena el texto del documento, sus tokens preprocesados, y un conteo de términos utilizando la estructura de datos `Counter` de la librería `collections`. Además, se implementaron métodos para la representación, iteración y acceso a los términos:

```
class Document:
    def __init__(self, text: str, name: str = 'nameless'):
        self.text = text
        self.name = name
        self.tokens = text_preprocessing(text)
        counter = Counter(self.tokens)
        self.term_counts = pd.Series(counter.values(), index=counter.keys())
```

4 Búsqueda Binaria usando Índice Invertido (BSII)

En esta sección se implementa un índice invertido, que es una estructura de datos ampliamente utilizada en motores de búsqueda para indexar y recuperar documentos. Posteriormente, se utiliza este índice para realizar búsquedas binarias, recuperando documentos que contienen todos los términos de una consulta dada.

4.1 Construcción del Índice Invertido

El índice invertido se construye mapeando cada término a una lista de documentos en los que aparece. A continuación se muestra cómo se puede implementar en Python.

```
class BSII:
    def __init__(self, docs: List[Document]):
```

```
        self.docs = docs
        self.inverse_index = {}

        for doc in self.docs:
            for term in doc.term_counts.index:
                if term not in self.inverse_index:
                    self.inverse_index[term] = set()
                self.inverse_index[term].add(doc)
```

4.2 Búsqueda Binaria en el Índice Invertido

La función de búsqueda se implementa para manejar consultas AND y NOT, permitiendo encontrar documentos que contienen todos los términos especificados o excluir documentos que contienen términos no deseados.

```
def search(self, query_document: Document = None, excluded_query_document: Document = None) ->
↪  set:
    relevant_docs = set(self.docs)

    if query_document is not None:
        for term in query_document.term_counts.index:
            if term in self.inverse_index:
                relevant_docs.intersection_update(self.inverse_index[term])

    if excluded_query_document is not None:
        for term in excluded_query_document.term_counts.index:
            if term in self.inverse_index:
                relevant_docs.difference_update(self.inverse_index[term])

    return relevant_docs
```

4.3 Evaluación de Consultas y Resultados

Finalmente, se evalúan las consultas y se guardan los resultados en el archivo indicado.

```
def evaluate_search(self, queries: List[Document], output_path: Path):
    with open(output_path, 'w') as output_file:
        for query in queries:
            relevant_docs = self.search(query_document=query)
            output_file.write(f"{query.name}\t{','.join(doc.name for doc in
            ↪  relevant_docs)}\n")
```

4.4 Resultados de las Queries

A continuación se presentan los resultados obtenidos al ejecutar las queries sobre el índice invertido. Para cada consulta (qXX), se muestran los documentos relevantes encontrados.

Query	Documentos Relevantes
q01	Ninguno
q02	d293, d291
q03	d147, d283, d105, d291, d318, d152
q04	d286
q06	d329, d026, d029, d303, d069, d257, d297
q07	d034, d004
q08	d117, d110, d251, d108, d205
q09	d198, d223, d205
q10	d231
q12	d277, d250
q13	Ninguno
q14	Ninguno
q16	d132, d150, d184, d277, d250, d229, d176
q17	d121, d271
q18	d203, d194, d210, d192
q19	d179
q22	Ninguno
q23	Ninguno
q24	d129, d240, d282, d221
q25	d020, d023, d245, d166, d211, d247, d156, d128, d167, d328, d265, d032
q26	Ninguno
q27	Ninguno
q28	d174, d136
q29	d037, d046, d294
q32	d025, d031, d254, d139, d090
q34	Ninguno
q36	d265, d257
q37	d169
q38	Ninguno
q40	Ninguno
q41	d150, d174
q42	Ninguno
q44	Ninguno
q45	d105
q46	d133, d094

Table 1: Resultados de las queries ejecutadas en el índice invertido.

En los resultados obtenidos, se puede observar que ciertas consultas no retornaron ningún documento relevante, lo que indica que los términos de búsqueda de esas queries no se encuentran en el índice invertido. Otras consultas retornaron múltiples documentos, lo que demuestra la capacidad del índice invertido para realizar búsquedas booleanas eficientes.

5 Recuperación de Documentos basada en Vectores (RRDV)

En esta sección se implementa el modelo de Recuperación de Documentos basada en Vectores (RRDV). Este modelo utiliza la representación TF-IDF (Frecuencia de Términos-Inversa Frecuencia de Documentos) para evaluar la similitud entre documentos. La similitud se mide utilizando la similitud coseno, que permite identificar documentos relevantes para una consulta dada basándose en la cercanía de sus vectores de características.

5.1 Construcción del Índice TF-IDF

La clase `RRDV` se inicializa con una lista de documentos y construye un índice TF-IDF para cada término en los documentos. La construcción del índice TF-IDF implica varios pasos clave:

1. **Cálculo de la Frecuencia de Términos (TF):** Se calcula la frecuencia de cada término en cada documento, lo que proporciona una medida básica de cuán relevante es un término dentro de un documento específico.
2. **Cálculo de la Inversa Frecuencia de Documentos (IDF):** Se calcula la IDF, que mide la rareza de un término en el corpus total de documentos. Los términos que aparecen en muchos documentos tienen un menor valor de IDF, mientras que los términos raros tienen un mayor valor de IDF.
3. **Construcción del Índice TF-IDF:** Finalmente, se combina el TF y el IDF para obtener el índice TF-IDF, que pondera los términos de manera que los términos comunes en todo el corpus reciben menos peso, mientras que los términos específicos de documentos individuales reciben más peso.

A continuación se muestra la implementación de esta parte del proceso:

```
class RRDV:
    def __init__(self, docs: List[Document]):
        # Crear un DataFrame con los conteos de términos para cada documento
        self.term_counts = pd.DataFrame({
            doc.name: doc.term_counts for doc in self.docs
        })
        self.term_counts.fillna(0, inplace=True)

        # Calcular la frecuencia de documentos para cada término
        self.document_count = (self.term_counts >= 1).sum(axis=1)

        # Calcular el IDF (Inverse Document Frequency)
        self.idf = np.log10(len(self.docs) / self.document_count)

        # Calcular TF-IDF (Term Frequency-Inverse Document Frequency)
        self.tfidf = np.log10(1 + self.term_counts).mul(self.idf, axis=0)
```

Justificación de la Estrategia

La estrategia de usar TF-IDF para representar documentos y consultas es ampliamente aceptada debido a su capacidad para capturar tanto la importancia local de un término en un documento (mediante la frecuencia de términos, TF) como su importancia global en el corpus (mediante la inversa frecuencia de documentos, IDF). Esta combinación permite que los términos que son significativos para el contenido de un documento, pero no son comunes en todo el corpus, tengan un mayor peso en el proceso de recuperación de información.

¿Es esta estrategia eficiente?

Desde una perspectiva computacional, el cálculo del TF-IDF es relativamente eficiente y escalable, especialmente cuando se manejan corpus grandes. El uso de estructuras de datos como `DataFrame` en pandas permite un manejo eficiente de los cálculos en términos de memoria y tiempo de procesamiento. Sin embargo, el proceso de calcular similitudes coseno para cada consulta contra todos los documentos en el corpus puede ser costoso en términos de tiempo, especialmente a medida que el tamaño del corpus crece. Esto es conocido como el problema de la escalabilidad en la recuperación de información.

Justificación de la Eficiencia:

- **Escalabilidad:** La estrategia es razonablemente escalable para corpora de tamaño moderado. Sin embargo, en aplicaciones a gran escala, podría ser necesario implementar optimizaciones adicionales, como el uso de índices invertidos, reducción de dimensionalidad, o técnicas de recuperación aproximada para manejar grandes volúmenes de datos eficientemente.
- **Eficiencia Espacial:** El uso de TF-IDF combinado con `DataFrame` permite una representación vectorial compacta y eficiente de los documentos, lo cual es crucial para evitar problemas de memoria cuando se trabaja con grandes colecciones de textos.
- **Eficiencia Temporal:** Aunque el cálculo de la similitud coseno es lineal con respecto al número de términos y documentos, el proceso puede volverse ineficiente para grandes volúmenes de datos. Por lo tanto, aunque la estrategia es eficiente para corpora de tamaño moderado, en escenarios con datos masivos se podría considerar el uso de técnicas de recuperación más avanzadas o escalables.

En resumen, la estrategia de usar TF-IDF y similitud coseno para la recuperación de documentos es efectiva y razonablemente eficiente para muchos escenarios. Sin embargo, para grandes volúmenes de datos, podrían ser necesarias optimizaciones adicionales para mantener la eficiencia temporal y espacial.

5.2 Búsqueda de Documentos por Similitud Coseno

El método `search` permite buscar documentos similares a una consulta, calculando la similitud coseno entre el vector TF-IDF de la consulta y los vectores de los documentos. Este proceso evalúa la relevancia de los documentos en función de su similitud con la consulta, devolviendo un conjunto de documentos ordenados por su similitud.

```
def search(self, query_document: Document, min_similarity: float = 0.0) -> pd.DataFrame:
    # Filtrar términos en el vocabulario del índice
    in_vocab_term_counts =
    ↪  query_document.term_counts[query_document.term_counts.index.isin(self.idf.index)]

    # Calcular el TF-IDF para el documento de consulta
    query_tfidf = (np.log10(1 + in_vocab_term_counts) * self.idf).fillna(0)

    # Calcular la similitud coseno entre el documento de consulta y todos los documentos en el
    ↪  índice
```

```
    similarities = self.cosine_similarity(query_tfidf, self.tfidf)

    # Crear un DataFrame con los resultados de similitud
    results = pd.DataFrame({
        'similarity': similarities,
        'doc': self.docs
    }, index=self.tfidf.columns)

    # Ordenar los resultados por similitud de mayor a menor
    results.sort_values(by='similarity', ascending=False, inplace=True)

    # Filtrar los resultados por similitud mínima
    results = results[results['similarity'] > min_similarity]

    return results
```

El enfoque basado en la similitud coseno es robusto para medir la similitud entre documentos representados en un espacio vectorial, haciendo de este un método confiable para la recuperación de información en bases de datos textuales.

5.3 Evaluación de las Consultas

El método `evaluate_search` permite evaluar un conjunto de consultas y escribir los resultados en un archivo de salida. Los documentos se ordenan por su similitud coseno con la consulta, y se guardan en un archivo para su posterior análisis.

```
def evaluate_search(self, queries: List[Document], output_path: Path):
    with open(output_path, 'w') as output_file:
        for query in queries:
            relevant_docs = self.search(query_document=query)
            result_texts = [f'{doc_name}:{row.similarity}' for doc_name, row in
            ↪  relevant_docs.iterrows()]
            output_file.write(f"{query.name}\t{','.join(result_texts)}\n")
```

5.4 Resultados de las Consultas

Los resultados de las consultas realizadas utilizando el modelo RRDV se presentan en un archivo separado. Este archivo contiene las similitudes coseno entre cada consulta y los documentos relevantes, ordenados de mayor a menor similitud.

El archivo con los resultados completos puede descargarse desde el siguiente enlace:

Descargar resultados de RRDV

5.5 Evaluación de Resultados con RRDV

Para el modelo de RRDV, se calcularon las métricas P@M, R@M y NDCG@M por cada consulta. M es el número de documentos relevantes encontrados en el archivo de juicios de relevancia por consulta. La métrica MAP, que es el promedio de las precisiones promedio (Average Precision) para todas las consultas, se calculó para proporcionar una evaluación general del rendimiento del modelo. Para el MAP del RRDV obtuvimos un resultado de:

```
MAP:  0.7476806265561247
```

Lo que indica un buen rendimiento en la recuperación de documentos relevantes para las consultas evaluadas.

A continuación, se muestran las distribuciones de las principales métricas: número de documentos relevantes (M), precisión en M (P@M), recall en M (R@M) y NDCG en M (nDCG@M).

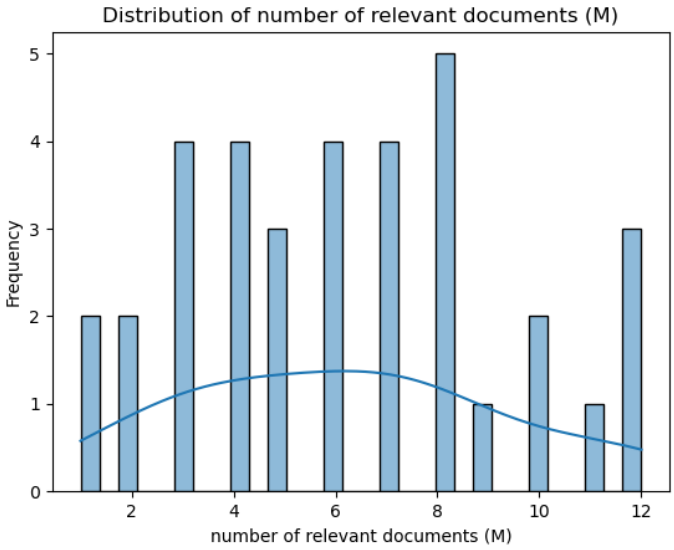

Figure 1: Distribución del número de documentos relevantes (M) para el modelo RRDV.

La Figura 1 muestra la distribución del número de documentos relevantes por consulta (M). Se observa que la mayoría de las consultas tienen entre 4 y 8 documentos relevantes. Esto sugiere que las consultas en nuestro conjunto de pruebas están diseñadas para identificar múltiples documentos relevantes, lo cual es esperado en escenarios donde se necesita una recuperación exhaustiva de información.

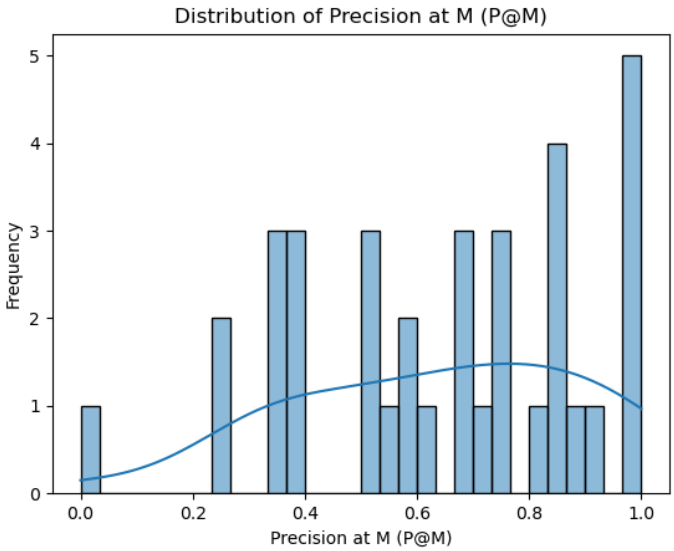

Figure 2: Distribución de la Precisión en M (P@M) para el modelo RRDV.

La Figura 2 muestra la distribución de la precisión en M (P@M). Se observa que muchas consultas alcanzan una precisión alta, con un número considerable de ellas logrando valores de precisión cercanos a 1. Esto indica que el modelo RRDV es efectivo en recuperar documentos relevantes dentro del conjunto de documentos relevantes M, aunque hay algunas consultas con una precisión considerablemente menor, lo que sugiere que el modelo podría estar recuperando algunos documentos irrelevantes.

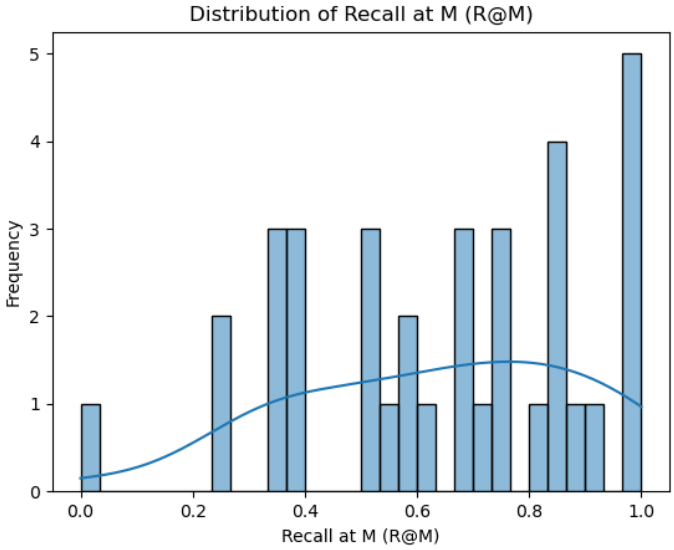

Figure 3: Distribución del Recall en M (R@M) para el modelo RRDV.

La Figura 3 presenta la distribución del recall en M (R@M). Aquí también se observa que muchas consultas logran un recall alto, con valores cercanos a 1. Esto implica que el modelo RRDV es capaz de recuperar la mayoría de los documentos relevantes en su conjunto. Sin embargo, existe una variabilidad significativa en las consultas, lo

que sugiere que algunas de ellas no logran recuperar todos los documentos relevantes.

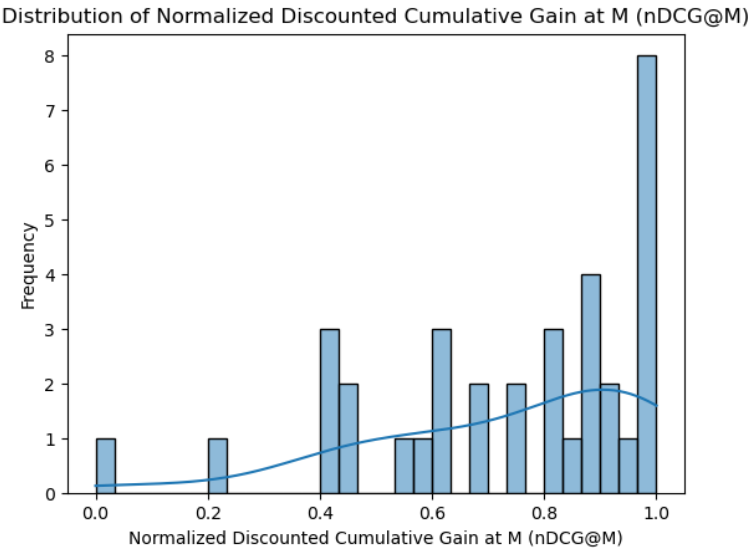

Figure 4: Distribución del NDCG en M (nDCG@M) para el modelo RRDV.

La Figura 4 muestra la distribución del NDCG en M (nDCG@M). Esta métrica toma en cuenta no solo la relevancia binaria de los documentos recuperados, sino también la relevancia no binaria, lo que ofrece una medida más refinada de la calidad de la recuperación. La distribución sugiere que, aunque hay un número significativo de consultas con un nDCG@M alto (cercano a 1), hay una dispersión más amplia en comparación con las métricas anteriores, indicando variaciones en la relevancia de los documentos recuperados.

Este análisis demuestra que el modelo RRDV funciona razonablemente bien en la mayoría de las consultas, logrando altos valores en precisión, recall y nDCG en M, aunque con alguna variabilidad que sugiere oportunidades de mejora en la recuperación de documentos relevantes.

6 Recuperación de Documentos mediante Vectores de Documentos con Gensim (RRDV con GENSIM)

Gensim es una biblioteca especializada en el procesamiento de lenguaje natural, que facilita la implementación de modelos como TF-IDF y la evaluación de similitudes entre documentos. En esta sección, se utiliza Gensim para construir un sistema de recuperación de información que utiliza la similitud coseno sobre vectores TF-IDF, simplificando así el proceso de RRDV.

6.1 Construcción del Índice TF-IDF con Gensim

La clase `GensimSearch` se inicializa con una lista de documentos y utiliza Gensim para construir un índice TF-IDF de manera eficiente. A continuación se muestra la implementación de esta parte del proceso.

```
class GensimSearch:
    def __init__(self, docs: List[Document]):
```

```
        # Implemente el concepto de un Dictionary, es decir, mapear palabras a enteros.
        self.dictionary = corpora.Dictionary(all_docs)

        # El corpus se construirá con el método doc2bow de gensim, que cuenta las ocurrencias
        ↪  de cada término en cada documento.
        self.corpus = [self.dictionary.doc2bow(doc) for doc in all_docs]

        # Construimos el modelo TF-IDF de Gensim con nuestro corpus.
        self.model = TfidfModel(self.corpus)

        # Calcula la similitud coseno para nuestro corpus y el modelo TF-IDF.
        self.index = similarities.MatrixSimilarity(self.model[self.corpus])
```

GENSIM facilita la creación de un diccionario que mapea términos a enteros y el cálculo del TF-IDF y la similitud coseno con funciones predefinidas, optimizando así el proceso.

6.2 Búsqueda de Documentos por Similitud Coseno usando Gensim

El método `search` en `GensimSearch` realiza la búsqueda de documentos similares a una consulta aplicando las funciones de Gensim para procesar la consulta y calcular la similitud coseno.

```
def search(self, query_document: Document, min_similarity: float = 0.0) -> pd.DataFrame:
    query_bow = self.dictionary.doc2bow(query_document)
    query_tfidf = self.model[query_bow]
    sims = self.index[query_tfidf]

    # Crear un DataFrame con los resultados de similitud
    results = pd.DataFrame({
        'similarity': sims,
        'doc': self.docs
    }, index=[doc.name for doc in self.docs])

    # Ordenar los resultados por similitud de mayor a menor
    results.sort_values(by='similarity', ascending=False, inplace=True)

    # Filtrar los resultados por similitud mínima
    results = results[results['similarity'] > min_similarity]

    return results
```

La consulta se procesa de manera similar a los documentos, convirtiéndola en un vector TF-IDF que luego se compara con los documentos indexados utilizando la similitud coseno.

6.3 Evaluación de las Consultas

El método `evaluate_search` permite evaluar un conjunto de consultas y escribir los resultados en un archivo de salida. Los documentos se ordenan por su similitud coseno con la consulta, y se guardan en un archivo para su posterior análisis.

```
def evaluate_search(self, queries: List[Document], output_path: Path):
    with open(output_path, 'w') as output_file:
        for query in queries:
            relevant_docs = self.search(query_document=query)
            result_texts = [f'{doc_name}:{row.similarity}' for doc_name, row in
            ↪  relevant_docs.iterrows()]
            output_file.write(f"{query.name}\t{','.join(result_texts)}\n")
```

6.4 Resultados de las Consultas con GENSIM

Los resultados de las consultas realizadas utilizando GENSIM se presentan en un archivo separado. Este archivo contiene las similitudes coseno entre cada consulta y los documentos relevantes, ordenados de mayor a menor similitud.

El archivo con los resultados completos puede descargarse desde el siguiente enlace:

Descargar Resultados de las Consultas con Gensim

6.5 Evaluación de Resultados con Gensim

Para el modelo implementado con Gensim, se siguió el mismo procedimiento de evaluación, calculando P@M, R@M y NDCG@M por consulta, y MAP como métrica general. El valor de MAP obtenido para el modelo fue de:

```
MAP:  0.7041615877565259
```

Aunque ligeramente inferior al obtenido con RRDV, este valor sigue reflejando un rendimiento competitivo.

A continuación, se muestran las distribuciones de las principales métricas: número de documentos relevantes (M), precisión en M (P@M), recall en M (R@M) y NDCG en M (nDCG@M).

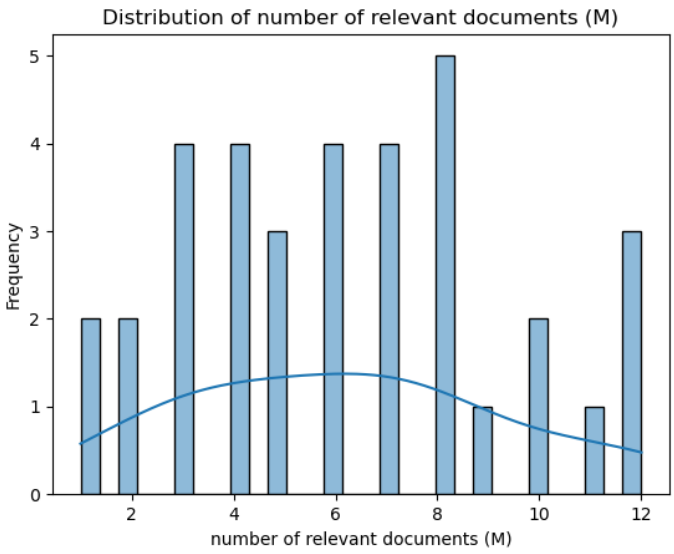

Figure 5: Distribución del número de documentos relevantes (M) para el modelo Gensim.

La Figura 5 muestra la distribución del número de documentos relevantes por consulta (M) para el modelo Gensim. Se observa que la distribución es similar a la del modelo RRDV, con la mayoría de las consultas teniendo entre 4 y 8 documentos relevantes. Esto sugiere que el modelo Gensim también identifica múltiples documentos relevantes para la mayoría de las consultas, cumpliendo con la necesidad de una recuperación exhaustiva de información.

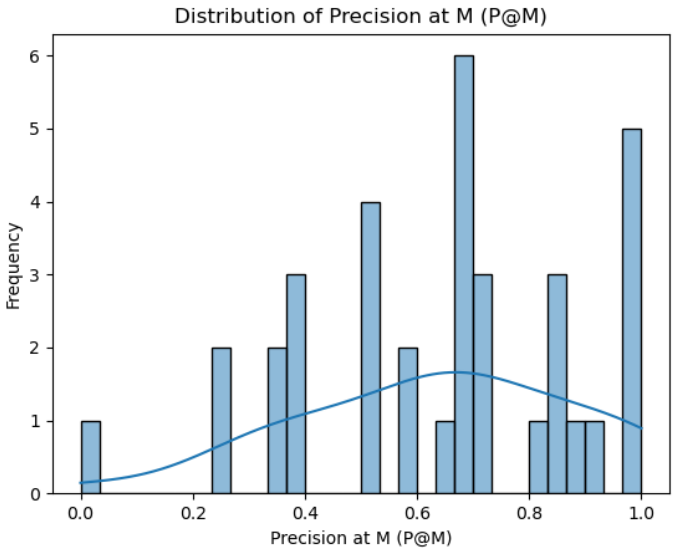

Figure 6: Distribución de la Precisión en M (P@M) para el modelo Gensim.

La Figura 6 muestra la distribución de la precisión en M (P@M) para el modelo Gensim. La precisión presenta una tendencia hacia valores altos, similar a RRDV, con picos notables en 0.7 y 1.0. Esto indica que Gensim es efectivo en recuperar documentos relevantes dentro del conjunto de documentos relevantes M, aunque hay cierta variabilidad que sugiere que en algunos casos el modelo podría no estar logrando una precisión óptima.

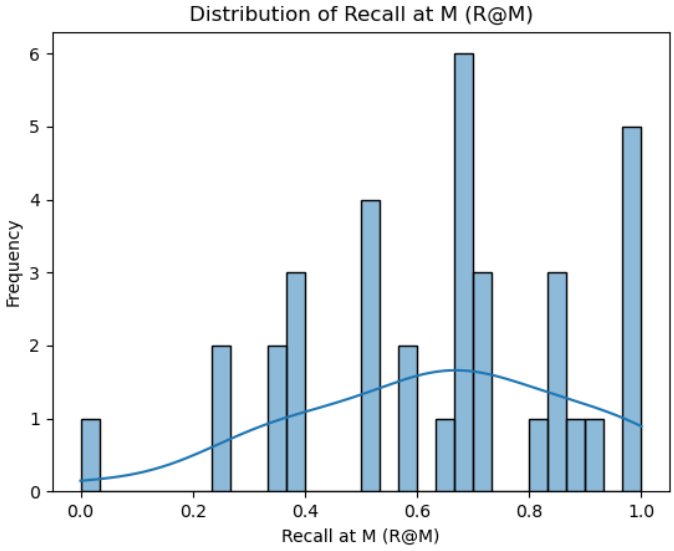

Figure 7: Distribución del Recall en M (R@M) para el modelo Gensim.

La Figura 7 presenta la distribución del recall en M (R@M). Se observa que muchas consultas alcanzan un recall alto, con valores cercanos entre 0.7 y 1. Esto implica que el modelo Gensim es capaz de recuperar la mayoría de los documentos relevantes en su conjunto, aunque, al igual que con la precisión, existe una variabilidad significativa, lo que sugiere que algunas consultas no logran recuperar todos los documentos relevantes.

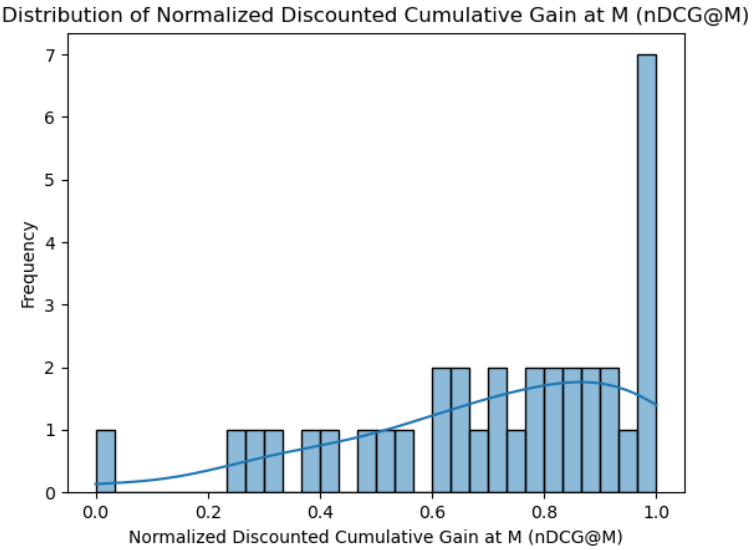

Figure 8: Distribución del NDCG en M (nDCG@M) para el modelo Gensim.

La Figura 8 muestra la distribución del NDCG en M (nDCG@M). Esta métrica, que considera la relevancia no binaria de los documentos, muestra una mayor dispersión en comparación con las métricas anteriores, con un sesgo hacia valores altos. Esto sugiere que, aunque Gensim es efectivo en la recuperación de documentos relevantes, puede haber variabilidad en la calidad del ordenamiento de estos documentos.

Este análisis demuestra que el modelo Gensim ofrece un rendimiento robusto en la mayoría de las consultas, alcanzando altos valores en precisión, recall y nDCG en M, aunque con una variabilidad que sugiere oportunidades de mejora en la consistencia del ordenamiento de los documentos relevantes.

7 Conclusión

En este trabajo, se han implementado y evaluado dos enfoques diferentes para la recuperación de información: el modelo RRDV y el modelo basado en Gensim. Ambos modelos se desarrollaron para realizar búsquedas eficientes utilizando un índice invertido y TF-IDF, permitiendo comparar su rendimiento a través de métricas estándar en el campo de la recuperación de información, como Precision at M (P@M), Recall at M (R@M) y NDCG at M (nDCG@M).

Los resultados obtenidos indican que ambos modelos son capaces de identificar y recuperar documentos relevantes con un alto grado de precisión y recall, lo que refleja su efectividad en escenarios de búsqueda exhaustiva. Sin embargo, se observó una cierta variabilidad en las métricas, especialmente en nDCG@M, que sugiere diferencias en cómo cada modelo prioriza y ordena los documentos relevantes.

El análisis de las distribuciones de las métricas revela que, aunque ambos modelos muestran un rendimiento robusto en la mayoría de las consultas, existen áreas donde se podría mejorar. En particular, la dispersión en nDCG@M indica que en algunos casos los modelos no logran optimizar completamente el orden de los documentos

recuperados en términos de su relevancia.

En términos de MAP, el modelo RRDV presentó un rendimiento ligeramente superior al modelo Gensim, lo cual podría estar relacionado con la implementación específica de TF-IDF en RRDV. No obstante, Gensim ofrece un enfoque más simplificado y automatizado, lo que podría ser ventajoso en aplicaciones donde la facilidad de implementación y la escalabilidad son prioritarias.

En resumen, ambos enfoques tienen fortalezas que los hacen adecuados para distintas aplicaciones de recuperación de información. El modelo RRDV ofrece una mayor precisión en el contexto de nuestro conjunto de datos y consultas, mientras que Gensim proporciona una implementación más accesible y eficiente, con un rendimiento competitivo en términos de las métricas evaluadas.