

Santiago Chica, 201620695. Andrés Mosquera, 202525003. David Briceño, 202526065.

1. Métricas de evaluación de IR usando python+numpy

- **precision** La definición clásica: $precisión2 = (\# \text{ de documentos relevantes recuperados}) / (\# \text{ total recuperados})$. Con np.array + np.sum se implementa el numerador y len(...) el denominador.
- **precision_at_k** Se usa “Precision at k ($P@k$)” para medir la calidad en los primeros k resultados. Cortar con [:k] y sumar con np.sum implementa tal cual $P@k = (\# \text{ relevantes en los } k \text{ primeros})/k$.
- **recall_at_k** Se define *recall* como $(\# \text{ relevantes recuperados})/(\# \text{ totales relevantes})$. El recall_at_k aplica **la misma definición** pero restringiendo a los primeros k (corte en k). Es la aplicación directa de la definición con un cutoff.
- **average_precision (AP)** En IR la *Average Precision* para una consulta es el **promedio de las precisiones** en cada posición donde aparece un documento relevante. El uso de np.where para localizar índices relevantes y el bucle que suma precision_at_k(..., index+1) implementa exactamente la ecuación.
- **mean_average_precision (MAP)** MAP es el promedio de las AP de todas las consultas. El bucle que acumula average_precision por consulta y divide por el número de consultas es justo la definición.
- **dcg_at_k** y **ndcg_at_k** Se define **DCG@p** como una suma descontada por la posición y se introduce dos variantes de ganancia: lineal (rel_i) y exponencial ($2^{\{rel_i\}} - 1$). Para **NDCG**, se divide el DCG por el **IDCG** (el DCG del ranking ideal), que se obtiene ordenando las relevancias de mayor a menor. — La función usa np.log2 para el descuento, np.sort(...)[::-1] para el ranking ideal y la **ganancia lineal** (Se implementa con $relevance / \log2(...)$). Variante exponencial como alternativa; con relevancia binaria ambas dan lo mismo.

2. Preprocesamiento

Conjunto de datos: hay tres archivos que componen el conjunto de datos.

- “Docs raw texts” contiene 331 documentos en formato NAF (XML; debe usar el título y el contenido para modelar cada documento).
- “Queries raw texts” contiene 35 consultas.
- “relevance-judgments.tsv” contiene para cada consulta los documentos considerados relevantes para cada una de las consultas. Estos documentos relevantes fueron construidos manualmente por jueces humanos y sirven como “ground-truth” y evaluación.

Para el preprocesamiento se realiza mediante los siguientes pasos:

Tokenización: Primer paso para obtener términos a partir del texto y poder construir un vocabulario.

Stopwords: Se eliminan porque aparecen con muchísima frecuencia y no ayudan a discriminar documentos (ej. *the, and, of*).

Normalización: Se eliminan caracteres no alfabéticos (números, caracteres especiales). Se eliminan acentos y letras que no hacen parte del alfabeto inglés.

Stemming: Agrupa variantes morfológicas de una palabra bajo una misma raíz, lo cual mejora la recuperación al reducir la dispersión del vocabulario. Se usa algoritmo de Porter.

Minúsculas: Todos los tokens se pasan a minúsculas, para evitar duplicados debido a minúsculas/mayúsculas.

3. Búsqueda binaria usando índice invertido (BSII) (numpy y pandas.)

3.1. Implementación del índice invertido usando los 331 documentos en el conjunto de datos.

Al almacenar el vocabulario en un dataframe de pandas podemos expandir los listados internos utilizando la función `.explode`, con ello podemos hacer una agrupación por los tokens en común entre documentos. Después se utiliza un diccionario para el almacenamiento y su futuro acceso. Además, se incluye la cantidad de frecuencia en el documento (que es el mismo largo de la lista). Se agrega una columna adicional con la frecuencia de término por documento. Esto será de utilidad al momento de calcular la matriz Tf-idf.

3.2. Función que lea el índice invertido y calcule consultas booleanas mediante el algoritmo de mezcla. El algoritmo de mezcla debe ser capaz de calcular: AND, y NOT.

El **procesamiento de consultas booleanas** se hace combinando listas de postings mediante el algoritmo de intersección (“merge algorithm”).

- **AND** = intersección de postings lists.
- **NOT** = diferencia con respecto al conjunto universal de documentos.

La función recorre postings y aplica operaciones de conjuntos para implementar la semántica booleana. Nos aprovechamos de la implementación nativa de los conjuntos en python para poder realizar directamente las operaciones, donde AND representa la intersección de conjuntos, OR representa la unión de conjuntos y NOT representa el complemento.

3.3. Para cada una de las 35 consultas en el conjunto de datos, recupere los documentos utilizando consultas binarias AND (i.e. *termino_1 AND termino_2 AND termino_3...*). Escriba un archivo (BSII-AND-queries_results) con los resultados siguiendo el mismo formato que "relevance-judgments": *q01 dXX,dYY,dZZ...*

Modelo booleano: permite consultas del tipo *t1 AND t2 AND t3 ...* que recuperan solo documentos que contienen **todos** los términos. Para la evaluación, se comparan los resultados contra los **juicios de relevancia** de un conjunto de pruebas (ground-truth).

Cada una de las 35 queries se procesa con **AND** porque se implementó el modelo booleano puro. El archivo de salida con el mismo formato que *relevance-judgments.tsv* es necesario para poder comparar los resultados del sistema con el ground-truth en las evaluaciones. Todo se hace siguiendo los pasos estándar en un sistema de recuperación de información (preprocesamiento → índice invertido → algoritmo de mezcla para consultas booleanas → resultados en formato comparable con relevance judgments).

4. Recuperación ranqueada y vectorización de documentos (RRDV) (numpy y pandas)

4.1. Función que, a partir del índice invertido, cree la representación vectorial ponderada tf.idf de un documento o consulta. Describa en detalle su estrategia, ¿es eficiente? ¿por qué sí, por qué no?

Estrategia:

1. TF se ajusta mediante logaritmos para mitigar el efecto de las frecuencias extremas.
2. IDF utiliza el logaritmo de la relación entre el total de documentos y el número de documentos que contienen el término, favoreciendo términos más raros.
3. El uso de un índice invertido facilita la eficiencia, pues permite acceder rápidamente a las frecuencias de los términos en los documentos y calcular la matriz de manera eficiente.

Es eficiente porque es un enfoque lógico y directo, ya que implementa el TF-IDF de manera adecuada siguiendo el concepto matemáticamente. No es eficiente en cuanto a tiempo y memoria para grandes volúmenes de datos. Ideas de optimización como evitar usar matrices dispersas, paralelización y pre-cálculo de dfi son pasos para hacer que la estrategia sea viable para una aplicación a gran escala.

4.2. Función que reciba dos vectores de documentos y calcule la similitud del coseno.

Se usa numpy para calcular la similitud del coseno, lo cual es una elección eficiente, ya que numpy realiza operaciones vectorizadas usando arrays. También se construye una función que calcula la similitud del coseno entre un vector de consulta y una matriz de documentos. Se implementa de esta forma para aprovechar las operaciones matriciales que ofrece numpy.

Se incluyó un control para evitar la división por cero, es decir, si alguno de los vectores tiene norma cero (lo cual indica que el vector no tiene términos representativos), el valor de similitud se asigna como 0.

4.3 Para cada una de las 35 consultas en el conjunto de datos, recupere los documentos clasificados -ordenados por el puntaje de similitud del coseno- (incluya solo los documentos con un puntaje superior a 0 para una consulta determinada). Escriba un archivo (RRDV-consultas_resultados) con los resultados siguiendo el siguiente formato:

q01 dXX: cos_simi(q01,dXX),dYY: cos_simi(q01, dYY),dZZ: cos_simi(q01,dZZ)...

Para consultar el conjunto de datos, se siguieron los siguientes pasos:

Vectorización de la consulta: La consulta se convierte en un vector de características. Para cada término en la consulta, se calcula su TF-IDF y luego se construye el vector query_vec. Este es un paso necesario para poder comparar la consulta con los documentos, usando las mismas representaciones.

Clasificación de documentos: Se usa la similitud del coseno para obtener una lista de documentos ordenados. La clasificación se realiza por el puntaje de similitud, de mayor a menor.

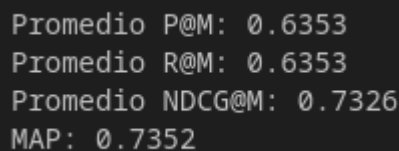
Filtrado de documentos: Solo se incluyen documentos con un puntaje superior a 0, para evitar almacenar o procesar documentos irrelevantes.

Formato de salida: Se usa un formato basado en consultas y documentos ordenados con el puntaje de similitud

4.4. Evaluación de resultados. Calcule $P@M$, $R@M$, $NDCG@M$ por consulta. M es el número de documentos relevantes encontrados en el archivo de juicios de relevancia por consulta. Luego calcule MAP como una métrica general.

NOTA I: Para $P@M$ y $R@M$ suponga una escala de relevancia binaria. Los documentos que no se encuentran en el archivo “relevance-judgments” NO son relevantes para una consulta determinada.

NOTA II: Para $NDCG@M$ utilice la escala de relevancia no binaria que se encuentra en el archivo “relevance-judgments”.



```
Promedio P@M: 0.6353
Promedio R@M: 0.6353
Promedio NDCG@M: 0.7326
MAP: 0.7352
```

Imagen 1. Resultados RRDV con numpy

El sistema tiene un equilibrio entre precisión y recall. Esto se puede interpretar como que el sistema no está sesgado por errores de Tipo I o Tipo II.

Áreas de mejora:

P@M y R@M podrían mejorarse si se optimizan las consultas, mejorando el preprocesamiento o utilizando otro método de puntaje de ranking de relevancia.

NDCG podría mejorar si se para asegura que los documentos relevantes sean siempre los primeros en el ranking.

5. Recuperación ranqueada y vectorización de documentos (RRDV) (GENSIM)

5.1. Función que, a partir del índice invertido, cree la representación vectorial ponderada tf.idf de un documento o consulta. Describa en detalle su estrategia, ¿es eficiente? ¿por qué sí, por qué no?

Se usa corpora.Dictionary y doc2bow para convertir los textos en representaciones de "Bolsa de Palabras". Transforma el texto en una forma que el modelo pueda entender y manipular.

Se aplica un modelo TF-IDF con models.TfidfModel, para representar la importancia relativa de las palabras en el corpus.

Se crea un índice de similitud usando similarities.SparseMatrixSimilarity. Permite calcular la similitud entre una consulta y todos los documentos.

5.2. Función que reciba dos vectores de documentos y calcule la similitud del coseno.

Se calcula la similitud utilizando la función similarities.SparseMatrixSimilarity de GENSIM. Esta implementación está diseñada para calcular la similitud de coseno entre documentos dispersos, como aquellos en formato bag-of-words, y un corpus de referencia, que en nuestro caso son los vectores de consulta. Su principal característica es que almacena la matriz de índice dispersa en la memoria RAM, utilizando la estructura scipy.sparse.csr. En caso de que el corpus exceda la memoria disponible, la implementación ofrece un control adicional a través del argumento maintain_sparsity, que permite elegir si el resultado se devuelve como una matriz dispersa o densa. Esta es una optimización en el uso de memoria frente a la implementación del punto 4.

5.3. Para cada una de las 35 consultas en el conjunto de datos, recupere los documentos clasificados -ordenados por el puntaje de similitud del coseno- (incluya solo los documentos con un puntaje superior a 0 para una consulta determinada). Escriba un archivo (RRDV-consultas_resultados) con los resultados siguiendo el siguiente formato:

$q01\ dXX: \cos_simi(q01, dXX), dYY: \cos_simi(q01, dYY), dZZ: \cos_simi(q01, dZZ)...$

Se usa la función descrita en el numeral anterior recorriendo cada uno de los vectores de consulta

5.4. Evaluación de resultados. Calcule $P@M$, $R@M$, $NDCG@M$ por consulta. M es el número de documentos relevantes encontrados en el archivo de juicios de relevancia por consulta. Luego calcule MAP como una métrica general.

NOTA I: Para $P@M$ y $R@M$ suponga una escala de relevancia binaria. Los documentos que no se encuentran en el archivo “relevance-judgments” NO son relevantes para una consulta determinada.

NOTA II: Para $NDCG@M$ utilice la escala de relevancia no binaria que se encuentra en el archivo “relevance-judgments”.

Comparación de Resultados:

	Métrica	RRDV	GENSIM	Diferencia (%)
0	P@M	0.6353	0.6273	1.275307
1	R@M	0.6353	0.6273	1.275307
2	NDCG@M	0.7326	0.7061	3.753009
3	MAP	0.7352	0.7014	4.818934

Tabla 1. Diferencia porcentual entre métricas RRDV GENSIM.

En la Tabla 1 podemos ver que el método RRDV implementado con funciones básicas de python obtuvo mejores métricas de recuperación que la implementación con GENSIM. Ahora bien, las métricas son similares, lo cual demuestra consistencia en el ejercicio. La principal diferencia está en la construcción de la matriz TF-IDF de ambas implementaciones.

Referencias

Manrique, R. (2025). Notas de clase ISIS-4221 Procesamiento de Lenguaje Natural. Universidad de los Andes. Bogotá, Colombia.