P3_modelos

April 6, 2021

Antonio Coín Castro
INE

1 Práctica 3: modelos de IR

Consideramos la siguiente matriz término-documento como descripción completa de una minicolección, correspondiente a la consulta q_1 de la práctica anterior. Suponemos que las palabras que tenemos aparecen 0 veces en el resto de documentos del corpus.

[1]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

%reload_ext autoreload
%autoreload 2
```

1.1 Lectura de datos

[2]:
```
# Load document data
df = pd.read_csv(
    "data/document-term.csv",
    header=0
)

# Save useful information
terms = list(df.columns[2:])
n_terms = len(terms)
D = df.shape[0]
rank_order = np.arange(1, D + 1, 1)
```

[3]:
```
# Load query data
q_text = "cuánto cobra jugador profesional fútbol sala"
q_words = q_text.split(' ')
```

1.2 Modelo vectorial

Determinar el ránking de los documentos para la consulta considerada en el modelo vectorial basado en *tf-idf* y coseno, es decir:

- Calcular los pesos de cada palabra en cada documento usando *tf-idf*.
- Utilizando los pesos calculados anteriormente, calcular el *score* de los documentos para la consulta utilizando similitud por ángulo.

En primer lugar calculamos los pesos $w(t, d)$ de cada palabra en cada documento, usando el siguiente esquema *tf-idf*.

Por un lado,

$$\mathrm{tf}(t, d) = \begin{cases} 1 + \log_2 \mathrm{freq}(t, d), & \text{si } \mathrm{freq}(t, d) > 0, \\ 0, & \text{en otro caso.} \end{cases}$$

y por otro,

$$\mathrm{idf(t)} = \log_2 \left(\frac{|\mathcal{D}| + 1}{|\mathcal{D}_t| + 0.5} \right),$$

donde $\mathcal{D}$ es la colección completa de documentos y $\mathcal{D}_t$ son los documentos que contienen el término t. Se introduce el suavizado para evitar resultados anómalos cuando $|\mathcal{D}_t| = 0$ ó $|\mathcal{D}_t| = |\mathcal{D}|$.

Si $\mathcal{V}$ es el vocabulario y llamamos $n = |\mathcal{V}|$, los vectores que representan los documentos en el espacio vectorial $\mathbb{R}^n$ tienen la forma:

$$\mathbf{d} = (d_1, \ldots, d_n), \quad \text{con } d_t = w(t, d) = \mathrm{tf}(t, d)\,\mathrm{idf}(t),\ t = 1, \ldots, n.$$

Por su parte, el vector que representa la consulta en $\mathbb{R}^n$ se calcula de manera análoga. Sin embargo, en este caso consideramos solo relevancia binaria (ya que no se reptien los términos en la consulta), y también consideramos conveniente omitir idf para no penalizar doblemente los términos muy comunes a la hora de calcular la similitud:

$$\mathbf{q} = (q_1, \ldots, q_n), \quad \text{con } q_t = \mathrm{tf}(t, q),\ t = 1, \ldots, n.$$

[4]:
```
# Compute tf and idf for the documents
tf = np.array([
    [1 + np.log2(df[term][d]) if df[term][d] > 0 else 0
     for d in range(D)] for term in terms])
idf = np.array([np.log2((D + 1)/(sum(df[term] > 0) + 0.5))
                for term in terms])

# Compute vectors d and q
docs = tf*idf[:, None]
q = np.array([1 if term in q_words else 0 for term in terms])
```

Para calcular la similitud $f(\mathbf{d}, \mathbf{q})$ de cada documento $\mathbf{d}$ frente a la consulta $\mathbf{q}$ basada en el coseno, hacemos lo siguiente:

$$f(\mathbf{d}, \mathbf{q}) = \frac{\mathbf{d} \cdot \mathbf{q}}{|\mathbf{d}|},$$

donde notamos que hemos eliminado el término $|\mathbf{q}|$, ya que no influye a la hora de hacer comparaciones.

[5]:
```
def score(d, q):
    return d@q/np.linalg.norm(d)


scores = np.array([score(d, q) for d in docs.T])
cosine_ranking = scores.argsort()[::-1]
scores = np.sort(scores)[::-1]

df_cosine = pd.DataFrame({
    'rank_cosine': rank_order,
    'id': cosine_ranking + 1,
    'score': scores}
)
df_cosine
```

[5]:

	rank_cosine	id	score
0	1	16	1.757432
1	2	15	1.277657
2	3	20	1.122590
3	4	3	1.030306
4	5	14	0.836464
5	6	11	0.822602
6	7	7	0.758935
7	8	8	0.702156
8	9	19	0.559610
9	10	13	0.513747
10	11	2	0.455264
11	12	6	0.439368
12	13	4	0.431580
13	14	12	0.414196
14	15	5	0.379693
15	16	1	0.326487
16	17	18	0.298897
17	18	9	0.275907
18	19	10	0.238879
19	20	17	0.218273

Como vemos, en este caso el documento que ha resultado el primero en el ránking es el 16, y el último el 17.

1.3 Modelos probabilísticos

Determinar el ránking de los documentos con algunos modelos probabilísticos a elección del alumno.

1.3.1 Modelo BIR

En el modelo Binary Independence Retrieval (BIR) se utilizan como base distribuciones de Bernoulli, y se ordenan los documentos por la probabilidad de relevancia. Tras varias simplificaciones e hipótesis de independencia, se llega a la *fórmula de Robertson-Spark-Jones (RSJ)*, dada por:

$$p(r \mid d, q) \propto RSJ(d, q) = \sum_{w \in d \cap q} \text{RSJ(w)} = \sum_{w \in d \cap q} \log \frac{p(w \mid r, q)}{1 - p(w \mid r, q)} + \log \frac{1 - p(w \mid \neg r, q)}{p(w \mid \neg r, q)}.$$

La estimación de parámetros dentro del modelo, en la que se introduce un suavizado de Laplace, es

$$p(w \mid r, q) \sim \frac{|r_w| + 0.5}{|R| + 1}, \quad p(w \mid \neg r, q) \sim \frac{|\mathcal{D}_w| - |r_w| + 0.5}{|\mathcal{D}| - |R| + 1},$$

donde:

- R es el conjunto de documentos relevantes,
- r_w es el conjunto de documentos relevantes que contienen la palabra w,
- D es el conjunto de todos los documentos considerados, y
- D_w es el conjunto de documentos que contienen a w.

El principal problema es que desconocemos $|R|$ y $|r_w|$, por lo que en una primera aproximación podemos considerar una muestra vacía donde ambos valen 0. Haciendo estas suposiciones y sustituyendo en la fórmula RSJ, obtenemos la siguiente expresión:

$$p(r \mid d, q) \propto \sum_{w \in d \cap q} \log \frac{|\mathcal{D}| - |\mathcal{D}_w| + 0.5}{|\mathcal{D}_w| + 0.5}.$$

Nota: suponemos que $|\mathcal{D}|$ es mucho mayor que el número de documentos que realmente tenemos, para evitar casos anómalos. Por ejemplo, establecemos $|\mathcal{D}| = 1000$.

[6]:
```
initial_scores = np.zeros(D)
D_ext = 1000

for w in q_words:
    Dw = np.sum(df[w] > 0)
    for d in range(D):
        if df[w][d] > 0:
            initial_scores[d] += np.log((D_ext - Dw + 0.5)/(Dw + 0.5))

initial_BIR_ranking = initial_scores.argsort()[::-1]
initial_scores = np.sort(initial_scores)[::-1]
```

```
df_BIR = pd.DataFrame({
    'rank_BIR': rank_order,
    'id': initial_BIR_ranking + 1,
    'score': initial_scores}
)
df_BIR
```

[6]:

	rank_BIR	id	score
0	1	8	23.807499
1	2	18	23.807499
2	3	2	23.807499
3	4	16	23.807499
4	5	15	23.807499
5	6	4	23.807499
6	7	13	23.807499
7	8	12	23.807499
8	9	11	23.807499
9	10	7	23.807499
10	11	1	19.888831
11	12	20	19.835169
12	13	19	19.778582
13	14	3	19.778582
14	15	17	19.778582
15	16	9	19.655189
16	17	6	19.655189
17	18	5	19.655189
18	19	10	19.655189
19	20	14	15.682860

Ahora podemos mejorar esta estimación inicial utilizando la técnica de *blind relevance feedback*. Elegimos por ejemplo $n = 7$ y consideramos como $\tilde{R}$ el conjunto de los top n documentos devueltos por la estimación inicial de $p(r \mid d, q)$. Utilizando este criterio de relevancia, podemos calcular los valores de $p(w \mid r, q)$ y $p(w \mid \neg r, q)$ que comentábamos arriba, y sustituirlos en la fórmula de RSJ, obteniendo la expresión siguiente:

$$p(r \mid d, q) \propto \sum_{w \in d \cap q} \log \frac{|\tilde{r}_w| + 0.5}{|\tilde{R}| - |\tilde{r}_w| + 0.5} + \log \frac{|\mathcal{D}| - |\mathcal{D}_w| + |\tilde{r}_w| - |\tilde{R}| + 0.5}{|\mathcal{D}_w| - |\tilde{r}_w| + 0.5}.$$

[7]:
```
def bir_rank(top_docs, R):
    scores = np.zeros(D)
    for w in q_words:
        Dw = np.sum(df[w] > 0)
        rw = np.sum(df[w][top_docs] > 0)
        for d in range(D):
            if df[w][d] > 0:
                scores[d] += (
                    np.log((rw + 0.5)/(R - rw + 0.5))
```

```
                    + np.log((D_ext - Dw + rw - R + 0.5)/(Dw - rw + 0.5)))
    ranking = scores.argsort()
    scores.sort()

    return ranking[::-1], scores[::-1]
```

[8]:
```
R = 7
top_docs = initial_BIR_ranking[:R]
BIR_ranking, scores = bir_rank(top_docs, R)

df_BIR_BRF = pd.DataFrame({
    'rank_BIR_BRF': rank_order,
    'id': BIR_ranking + 1,
    'score': scores}
)
df_BIR_BRF
```

[8]:

	rank_BIR_BRF	id	score
0	1	8	42.922979
1	2	18	42.922979
2	3	2	42.922979
3	4	16	42.922979
4	5	15	42.922979
5	6	4	42.922979
6	7	13	42.922979
7	8	12	42.922979
8	9	11	42.922979
9	10	7	42.922979
10	11	1	35.851576
11	12	20	35.767176
12	13	19	35.675187
13	14	3	35.675187
14	15	17	35.675187
15	16	9	35.461846
16	17	6	35.461846
17	18	5	35.461846
18	19	10	35.461846
19	20	14	28.306042

[9]:
```
if (BIR_ranking == initial_BIR_ranking).all():
    print("El nuevo ránking coincide con el inicial.")
else:
    print("El nuevo ránking es distinto al inicial.")
```

```
El nuevo ránking coincide con el inicial.
```

En este caso vemos que no cambia el ránking con respecto al inicial, por lo que nos hace pensar que podemos seleccionar un mayor valor de n. Escogemos por ejemplo $n = 15$ y probamos de nuevo.

[10]:
```
R = 15
top_docs = initial_BIR_ranking[:R]
BIR_ranking, scores = bir_rank(top_docs, R)

df_BIR_BRF = pd.DataFrame({
    'rank_BIR_BRF': rank_order,
    'id': BIR_ranking + 1,
    'score': scores}
)
df_BIR_BRF
```

[10]:

	rank_BIR_BRF	id	score
0	1	4	49.816854
1	2	18	49.816854
2	3	16	49.816854
3	4	15	49.816854
4	5	2	49.816854
5	6	13	49.816854
6	7	12	49.816854
7	8	11	49.816854
8	9	8	49.816854
9	10	7	49.816854
10	11	17	43.360573
11	12	19	43.360573
12	13	3	43.360573
13	14	1	42.364855
14	15	20	42.163165
15	16	5	38.796570
16	17	6	38.796570
17	18	9	38.796570
18	19	10	38.796570
19	20	14	31.142882

[11]:
```
if (BIR_ranking == initial_BIR_ranking).all():
    print("El nuevo ránking coincide con el inicial.")
else:
    print("El nuevo ránking es distinto al inicial.")
```

```
El nuevo ránking es distinto al inicial.
```

BONUS: Podemos hacer lo mismo pero considerando como conjunto pseudo-relevante el top n del ranking basado en *tf-idf* de la sección anterior. Esta vez podemos incrementar n de primera hora, ya que tenemos más confianza en la estimación. Tomamos por ejemplo $n = |\tilde{R}| = 15$.

[12]:
```
R = 15
top_docs = cosine_ranking[:R]
BIR_ranking, scores = bir_rank(top_docs, R)
```

```
df_BIR_BRF_cosine = pd.DataFrame({
    'rank_BIR_BRF_cosine': rank_order,
    'id': BIR_ranking + 1,
    'score': scores}
)
df_BIR_BRF_cosine
```

[12]:

	rank_BIR_BRF_cosine	id	score
0	1	7	46.905015
1	2	18	46.905015
2	3	2	46.905015
3	4	16	46.905015
4	5	15	46.905015
5	6	4	46.905015
6	7	13	46.905015
7	8	12	46.905015
8	9	11	46.905015
9	10	8	46.905015
10	11	20	40.035302
11	12	5	39.994712
12	13	6	39.994712
13	14	10	39.994712
14	15	9	39.994712
15	16	19	39.833612
16	17	3	39.833612
17	18	17	39.833612
18	19	1	38.086024
19	20	14	33.124999

[13]:
```
if (BIR_ranking == initial_BIR_ranking).all():
    print("El nuevo ránking coincide con el inicial.")
else:
    print("El nuevo ránking es distinto al inicial.")
```

```
El nuevo ránking es distinto al inicial.
```

BONUS 2: En lugar de usar ningún ránking que hayamos calculado, utilizamos los juicios de relevancia de la práctica anterior para establecer los documentos relevantes. En este caso, utilizamos todos los documentos con relevancia positiva, convenientemente ordenados.

[14]:
```
# Get relevant documents and order them by relevance
qrels = pd.read_csv(
    "../P2/data/qrels.csv",
    usecols=[0, 3],
    header=0,
    names=["id", "relevance"]
)
qrels = qrels[(qrels['id'] == 1) & (qrels['relevance'] > 0)]
```

```
qrels_ranking_rel = qrels['relevance'].argsort()[::-1]

R = len(qrels_ranking_rel)
BIR_ranking, scores = bir_rank(qrels_ranking_rel, R)

df_BIR_BRF_qrels = pd.DataFrame({
    'rank_BIR_BRF_qrels': rank_order,
    'id': BIR_ranking + 1,
    'score': scores}
)
df_BIR_BRF_qrels
```

[14]:

	rank_BIR_BRF_qrels	id	score
0	1	18	39.478305
1	2	2	39.478305
2	3	16	39.478305
3	4	15	39.478305
4	5	4	39.478305
5	6	13	39.478305
6	7	12	39.478305
7	8	11	39.478305
8	9	8	39.478305
9	10	7	39.478305
10	11	10	33.882968
11	12	6	33.882968
12	13	5	33.882968
13	14	9	33.882968
14	15	1	33.506533
15	16	19	33.330142
16	17	3	33.330142
17	18	17	33.330142
18	19	20	32.106367
19	20	14	26.511030

[15]:
```
if (BIR_ranking == initial_BIR_ranking).all():
    print("El nuevo ránking coincide con el inicial.")
else:
    print("El nuevo ránking es distinto al inicial.")
```

```
El nuevo ránking es distinto al inicial.
```

1.3.2 Modelo BM25

Este modelo probabilístico se ayuda de la fórmula RSJ para estimar para cada documento una función de *score*, cuya expresión es:

$$f(d,q) = \sum_{w \in q} \frac{(k+1)\text{freq}_{w,d}}{k(1-b+b|d|/\bar{d})) + \text{freq}_{w,d}} RSJ(w)$$

Los parámetros y las variables son:

- $|d|$ es la longitud del documento d (en nuestro caso suponemos que es la suma de las palabras que aparecen en el documento y están en nuestra colección),
- $\bar{d}$ es la media de las longitudes de todos los documentos $d \in \mathcal{D}$, es decir, $\bar{d} = |\mathcal{D}|^{-1} \sum_{d'} |d'|$,
- $b \in [0, 1]$ es un parámetro libre, y
- $k \geq 0$ es otro parámetro libre.

Aunque este modelo tiene una expresión muy sencilla y fácil de implementar, elegir los parámetros b y k no es fácil en general. Se ha observado mediante experimentación que $b = 0.75$ y $k \in [1.2, 2]$ funcionan bien.

[16]:
```
def RSJ(w, Dw):
    return np.log((D_ext - Dw + 0.5)/(Dw + 0.5))


def long(d):
    return np.sum(df.iloc[d][2:])


scores = np.zeros(D)
d_mean = np.mean([long(d) for d in range(D)])
k = 1.5
b = 0.75

for w in q_text.split():
    Dw = np.sum(df[w] > 0)
    for d in range(D):
        scores[d] += (((k + 1)*df[w][d]) /
                      (k*(1 - b + b*long(d)/d_mean) + df[w][d]))*RSJ(w, Dw)

BM25_ranking = scores.argsort()[::-1]
scores = np.sort(scores)[::-1]

df_BM25 = pd.DataFrame({
    'rank_BM25': rank_order,
    'id': BM25_ranking + 1,
    'score': scores}
)
df_BM25
```

[16]:

	rank_BM25	id	score
0	1	15	49.455436
1	2	16	49.093328
2	3	11	45.314359
3	4	2	43.062105
4	5	4	42.244836
5	6	12	42.152447

```
6           7  13  41.985631
7           8   8  41.983811
8           9   6  39.932617
9          10   3  39.693757
10         11   5  38.672495
11         12  20  38.424117
12         13   7  38.333674
13         14   1  37.769316
14         15   9  36.937988
15         16  18  36.366801
16         17  19  35.717794
17         18  10  35.631258
18         19  17  31.141127
19         20  14  30.990047
```

1.4 Modelo QLJM

Por último, implementamos el modelo de lenguaje Query Likelihood con suavizado de Jelinek-Mercer, donde ahora la función de ránking es proporcional a una estimación de $\log p(q \mid d)$. La expresión de este modelo para un $\lambda \in [0, 1]$ es:

$$\log p(q \mid d) \propto \sum_{w \in q} \log \left(1 + \frac{1-\lambda}{\lambda} \cdot \frac{\text{freq}_{w,d}}{|d|} \cdot \frac{\sum_{d'} |d'|}{\sum_{d'} \text{freq}_{w,d'}} \right).$$

Para nuestra prueba elegimos por ejemplo $\lambda = 0.5$.

[17]:
```python
def long(d):
    return np.sum(df.iloc[d][2:])

scores = np.zeros(D)
d_sum = np.sum([long(d) for d in range(D)])
lamb = 0.5

for w in q_text.split():
    freq_sum = np.sum([df[w][d] for d in range(D)])
    for d in range(D):
        scores[d] += np.log(
            1 + ((1-lamb)*df[w][d]*d_sum)/(lamb*long(d)*freq_sum)
        )

QLJM_ranking = scores.argsort()[::-1]
scores = np.sort(scores)[::-1]

df_QLJM = pd.DataFrame({
    'rank_QLJM': rank_order,
    'id': QLJM_ranking + 1,
    'score': scores}
```

```
)
df_QLJM
```

[17]:

	rank_QLJM	id	score
0	1	16	5.530966
1	2	15	5.226853
2	3	11	4.202748
3	4	13	4.168279
4	5	5	3.900055
5	6	14	3.844928
6	7	4	3.840355
7	8	6	3.758660
8	9	20	3.698705
9	10	3	3.694073
10	11	2	3.633478
11	12	8	3.626587
12	13	9	3.522384
13	14	1	3.501753
14	15	7	3.417964
15	16	19	3.387771
16	17	12	3.300617
17	18	10	2.923917
18	19	18	2.667670
19	20	17	2.581951

1.5 Comparación de los modelos

Unimos todos los ránkings obtenidos en una tabla para compararlos entre sí.

[18]:
```
all_models = [
    df_cosine.rename(columns={"id": "tf-idf"})['tf-idf'],
    df_BIR.rename(columns={"id": "BIR"})['BIR'],
    df_BIR_BRF.rename(columns={"id": "BIR_BRF"})['BIR_BRF'],
    df_BIR_BRF_cosine.rename(
        columns={"id": "BIR_BRF_cosine"})['BIR_BRF_cosine'],
    df_BIR_BRF_qrels.rename(
        columns={"id": "BIR_BRF_qrels"})['BIR_BRF_qrels'],
    df_BM25.rename(columns={"id": "BM25"})['BM25'],
    df_QLJM.rename(columns={"id": "QLJM"})['QLJM']
]
df_all = pd.concat(all_models, axis=1)
df_all
```

[18]:

	tf-idf	BIR	BIR_BRF	BIR_BRF_cosine	BIR_BRF_qrels	BM25	QLJM
0	16	8	4	7	18	15	16
1	15	18	18	18	2	16	15
2	20	2	16	2	16	11	11
3	3	16	15	16	15	2	13

4	14	15	2	15	4	4	5
5	11	4	13	4	13	12	14
6	7	13	12	13	12	13	4
7	8	12	11	12	11	8	6
8	19	11	8	11	8	6	20
9	13	7	7	8	7	3	3
10	2	1	17	20	10	5	2
11	6	20	19	5	6	20	8
12	4	19	3	6	5	7	9
13	12	3	1	10	9	1	1
14	5	17	20	9	1	9	7
15	1	9	5	19	19	18	19
16	18	6	6	3	3	19	12
17	9	5	9	17	17	10	10
18	10	10	10	1	20	17	18
19	17	14	14	14	14	14	17

Podemos ver que los rankings tienen cierta similaridad. Por ejemplo, el último documento es siempre el 14 o el 17, y el documento 16 está casi siempre de los primeros. Sin embargo, observamos variaciones en el ránking dependiendo del modelo elegido, lo que nos hace pensar que cada uno tiene su área de aplicación y que no hay uno que sobresalga frente a los demás. Incluso dentro de las variaciones del modelo BIR encontramos varias diferencias, si bien la tónica general del ránking se mantiene.

1.6 Evaluación y comparación de los modelos

Utilizando los juicios de relevancia de la práctica 2, comparar la efectividad de los diferentes modelos (entre sí y con los buscadores comerciales) con una métrica a elección del estudiante.

Para comparar los modelos elegimos una métrica triple: consideraremos la precisión, el recall y la media armónica. Comenzamos recalculando estos valores para la consulta q_1 utilizando los ránkings de los buscadores comerciales de la práctica anterior.

[19]:
```
# Load results and qrels

rels_engine = pd.read_csv(
    "../P2/data/rels_engine.csv",
    usecols=[0, 2, 5],
    header=0,
    names=["id", "url", "engine"]
)
rels_engine = rels_engine[(rels_engine['id'] == 1)]

qrels = pd.read_csv(
    "../P2/data/qrels.csv",
    usecols=[0, 2, 3],
    header=0,
    names=["id", "url", "relevance"]
```

```
)
qrels = qrels[(qrels['id'] == 1)]

# Cross join of tables
df_eng = rels_engine.merge(qrels, on=['id', 'url'], how='outer')
df_eng = df_eng.drop(['url', 'id'], axis=1)

# Sort dataframe
df_eng = df_eng.sort_values(by=["engine"])
```

[20]:
```
# Save useful information
engines = df_eng['engine'].unique()
M = len(engines)
AT = 10  # metrics @10
```

[21]:
```
P_eng = np.zeros(M)
R_eng = np.zeros(M)
F_eng = np.zeros(M)

total_relevant = sum(qrels['relevance'] > 0)

for j in range(M):
    relevant_eng = sum(
        (df_eng['engine'] == engines[j])
         & (df_eng['relevance'] > 0)
    )

    # Precision
    P_eng[j] = relevant_eng/AT

    # Recall
    R_eng[j] = relevant_eng/total_relevant

    # Harmonic mean
    F_eng[j] = 2*P_eng[j]*R_eng[j] / \
        (P_eng[j] + R_eng[j]) if relevant_eng > 0 else 0.0
```

Para tf-idf y los método probabilísticos, cogemos únicamente los 10 primeros resultados del ránking, para poder comparar de forma justa con los buscadores (de los que solo teníamos 10 resultados en cada uno). De esta forma también evitamos falsear las métricas, pues si cogiésemos todos los documentos obtendríamos siempre los mismos resultados de las mismas, ya que siempre tendríamos todos los documentos relevantes.

[23]:
```
method_names = ["Tf-idf", "BIR", "BIR+BRF", "BM25", "QLJM"]
all_models_df = [
    df_cosine,
    df_BIR,
    df_BIR_BRF,
```

```
    df_BM25,
    df_QLJM
]

K = len(method_names)
P_methods = np.zeros(K)
R_methods = np.zeros(K)
F_methods = np.zeros(K)

AT=5
total_relevant = sum(qrels['relevance'] > 0)
qrels_rel = qrels[qrels['relevance'] > 0]

for j, df_m in enumerate(all_models_df):
    df_join = df_m.merge(df, on=['id'], how='outer')['url']
    relevant_method = 0.0

    for url in df_join[:AT]:
        if url in qrels_rel['url'].tolist():
            relevant_method += 1

    # Precision
    P_methods[j] = relevant_method/AT

    # Recall
    R_methods[j] = relevant_method/total_relevant

    # Harmonic mean
    F_methods[j] = 2*P_methods[j]*R_methods[j] / \
        (P_methods[j] + R_methods[j]) if relevant_method > 0 else 0.0
```

Mostramos finalmente la comparación de nuestros métodos, entre sí y con los buscadores comerciales.

[24]:
```
methods = np.concatenate((engines, method_names))
P = np.concatenate((P_eng, P_methods))
R = np.concatenate((R_eng, R_methods))
F = np.concatenate((F_eng, F_methods))

eval_df = pd.DataFrame(
    data=np.array([methods, P, R, F]).T,
    columns=["Method/engine", "Precision", "Recall", "F"]
)

eval_df
```

[24]:

	Method/engine	Precision	Recall	F
0	bing	0.4	0.5	0.444444
1	duckduckgo	0.3	0.375	0.333333
2	ecosia	0.4	0.5	0.444444
3	google	0.7	0.875	0.777778
4	Tf-idf	0.6	0.375	0.461538
5	BIR	0.6	0.375	0.461538
6	BIR+BRF	0.8	0.5	0.615385
7	BM25	0.8	0.5	0.615385
8	QLJM	0.4	0.25	0.307692

Como vemos, todos los modelos nuevos superan a todos los buscadores excepto a Google, que de nuevo se impone como el claro vencedor en todas las métricas. Dentro de los modelos implementados en esta práctica, los que mejores resultados obtienen son BIR con *blind relevance feedback* y BM25. Como conclusión observamos que eligiendo el modelo probabilístico adecuado podemos llegar a superar a algunos buscadores comerciales, si bien la muestra que tenemos es pequeña y no necesariamente representativa del comportamiento global.