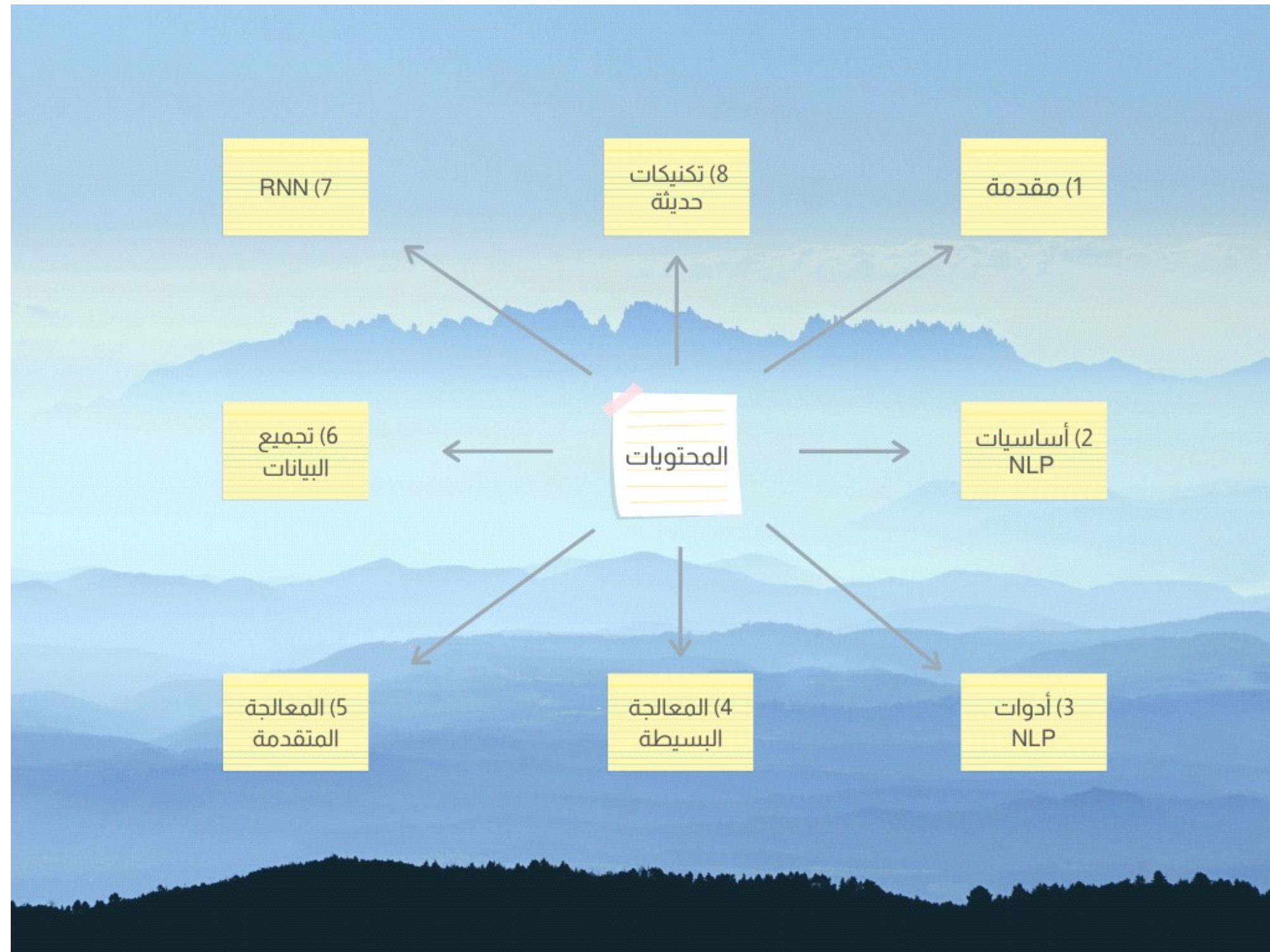


# NATURAL LANGUAGE PROCESSING

# المعالجة اللغوية الطبيعية



# المحتويات

|                 |                          |                 |                       |             |                        |            |                |                   |                      |
|-----------------|--------------------------|-----------------|-----------------------|-------------|------------------------|------------|----------------|-------------------|----------------------|
|                 |                          |                 |                       | التطبيقات   | العقبات و التحديات     | تاريخ NLP  | ما هو NLP      | المحتويات         | 1) مقدمة             |
|                 |                          |                 |                       |             | البحث في النصوص        | ملفات pdf  | الملفات النصية | المكتبات          | 2) أساسيات NLP       |
| T.Visualization | Syntactic Struc.         | Matchers        | Stopwords             | NER         | Stem & Lemm            | POS        | Sent. Segm.    | Tokenization      | 3) أدوات NLP         |
|                 | Dist. Similarity         | Text Similarity | TF-IDF                | BOW         | Word2Vec               | T. Vectors | Word embed     | Word Meaning      | 4) المعالجة البسيطة  |
| T. Generation   | L. Modeling              | NGrams          | Lexicons              | GloVe       | NMF                    | LDA        | T. Clustering  | T. Classification | 5) المعالجة المتقدمة |
|                 | Summarization & Snippets |                 | Ans. Questions        |             | Auto Correct           | Vader      | Naïve Bayes    | Sent. Analysis    |                      |
| Search Engine   | Relative Extraction      |                 | Information Retrieval |             | Information Extraction |            | Data Scraping  | Tweet Collecting  | 6) تجميع البيانات    |
|                 |                          |                 |                       |             | Rec NN\TNN             | GRU        | LSTM           | Seq to Seq        | 7) RNN               |
| Chat Bot        | Gensim                   | FastText        | Bert                  | Transformer | Attention Model        | T. Forcing | CNN            | Word Cloud        | 8) تكنيكات حديثة     |

## القسم السادس : تجميع البيانات

### الجزء السادس : Search Engine



تعد محركات البحث من أهم تطبيقات NLP , حيث تحرص المحركات علي تطوير أدواتها و تقنياتها صورة مستمرة , حتي تلبي احتياجات المستخدمين

و تعد أهم 3 سمات في أي محرك بحث :

- ايجاد النتائج بدقة
- حذف النتائج غير المناسبة
- سرعة عملية البحث



لكن ما هي معضلات البحث عن جمل ؟

فلو قام شخص بالبحث عن جملة من كلمتين مثل Stanford university فالمشكلة ان المحرك قد يظهر له نتائج قد لا تكون غير مرتبطة , مثل جملة " I went to university in Stanford " و هذه مشكلة , لان هذه الجملة تتطابق مع الكلمتين اللذان بحثنا عنهما , لكنها نتيجة غير مقبولة اذن المشكلة في ان حفظ وجود الكلمة في مستند ما فقط , هو غير كافي

## Phrase queries

- We want to be able to answer queries such as ***"stanford university"*** – as a phrase
- Thus the sentence *"I went to university at Stanford"* is not a match.
  - The concept of phrase queries has proven easily understood by users; one of the few "advanced search" ideas that works
  - Many more queries are *implicit phrase queries*
- For this, it no longer suffices to store only *<term : docs>* entries

هناك طريقتين لحل هذه المشكلة

الطريقة الأولى هي : ترقيم ثنائي الكلمة

و هي التي تعني ان يتم تخزين كل كلمتين متعاقبتين معا , و ليس كلمة واحدة

فلو قمنا بالبحث عن friends roman فنتمكن من ايجادها اذا كانت الكلمتين متعاقبتين

## A first attempt: Biword indexes

- Index every consecutive pair of terms in the text as a phrase
- For example the text “Friends, Romans, Countrymen” would generate the biwords
  - *friends romans*
  - *romans countrymen*
- Each of these biwords is now a dictionary term
- Two-word phrase query-processing is now immediate.

و وقتها اذا كان هناك بحث عن عدد اكبر من الكلمات , فيمكن معالجته عبر and , فجملة :  
Stanford university palo alto

وقتها نقوم بالبحث عن كلمتي Stanford university و university palo و palo alto , علي ان يكون بينهم  
and و هنا سيتم البحث بدقة

## Longer phrase queries

- Longer phrases can be processed by breaking them down
- **stanford university palo alto** can be broken into the Boolean query on biwords:

**stanford university AND university palo AND palo alto**

Without the docs, we cannot verify that the docs matching the above Boolean query do contain the phrase.

Can have false positives!

و لكن هذه الطريقة لها العديد من العيوب , ابرزها البطء الشديد , و ايضا انها غير مناسبة لو كنا نبحث عن كلمتين بينهما عدد معين من الكلمات , مثلا :

I was living in NYC

فالبحت عن كلمتي NYC , I بينهما 3 كلمات سيكون عسير

## Issues for biword indexes

---

- False positives, as noted before
- Index blowup due to bigger dictionary
  - Infeasible for more than biwords, big even for them
- Biword indexes are not the standard solution (for all biwords) but can be part of a compound strategy



الحل الثاني الأدق , هو الترقيم الموضعي

و يقصد به الا يتم فقط ترقيم ان كلمة كذا موجودة في مستند كذا

ولكن , ان يتم اولا تخزين الكلمة , و عدد مرات تكرارها

ثم يتم سرد رقم المستند , ثم : , ثم مواضعها في الملف نفسه , و هكذا

## Solution 2: Positional indexes

- In the postings, store, for each **term** the position(s) in which tokens of it appear:

<**term**, number of docs containing **term**;

*doc1*: position1, position2 ... ;

*doc2*: position1, position2 ... ;

etc.>



فكلمة be تواجدت في مستند رقم 1 في مكان كذا و كذا , ومستند رقم 2 في مكان كذا و كذا

## Positional index example

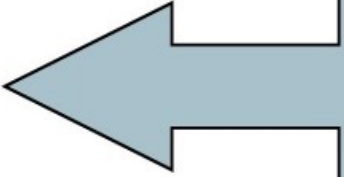
<*be*: 993427;

*1*: 7, 18, 33, 72, 86, 231;

*2*: 3, 149;

*4*: 17, 191, 291, 430, 434;

*5*: 363, 367, ...>



Which of docs *1,2,4,5* could contain "*to be or not to be*"?

- For phrase queries, we use a merge algorithm recursively at the document level
- But we now need to deal with more than just equality

الان اذا اردنا البحث عن جملة كاملة مثل "to be or not to be"

نقوم اولاً بايجاد ملفي كلمة *to* , *be* , ثم استبعاد المستندات التي ظهرت في كلمة واحدة ( مستند 1 و 2 و 5 و 7 )

نتناول مستند 4 . , نحن نعلم ان كلمة *be* جاءت تالية لـ *to* , لذا فيجب ان يكون موضعها رقم تالي لكلمة *to*

فاننا نتناول لكلمة *to* ارقام 16 , 190 , 429 , 433 , لكن نحن نعلم ان كلمة *to* جاءت مرة اخري بعد *to* الاولى بينهما 3 كلمات لذا فالرقم الوحيد هو 429 , و يمكن تطبيق هذه الفكرة لاي جملة اخري

## Processing a phrase query

- Extract inverted index entries for each distinct term:  
***to, be, or, not.***
- Merge their *doc:position* lists to enumerate all positions with “***to be or not to be***”.
  - ***to:***
    - 2:1,17,74,222,551; 4:8,16,190,429,433; 7:13,23,191; ...
  - ***be:***
    - 1:17,19; 4:17,191,291,430,434; 5:14,19,101; ...
- Same general method for proximity searches



## نتناول نموذج : البحث ذو الدرجات للمعلومات Ranked Retrieval Search

لفهم فكرة البحث ذو الدرجات , علينا اولاً ان نتعرف علي عيوب النظام الذي درسناه الان , وهو البحث عن كلمة معينة , ويمكن ان نسميه : البحث الثنائي Boolean retrieval

و مشكلة البحث الثنائي , انه ذو قيمة 1 او 0 , اي ان النتيجة اما تأتي او لا تأتي

و هنا توجد مشكلتين اساسيتين . . اما ان تكون كلمات البحث غير موجودة , فتكون النتيجة 0 , او تكون موجودة , وتأتي النتيجة بعشرات الالاف , و التي غالبها قد لا يكون متعلقاً بالمطلوب

فقد يقوم شخص بالبحث عن كلمات "الهجرة الي كنده" , مع كتابة حروف خاطئة , فتأتي النتيجة صفر , او ان يكتب حروف صحيحة "الهجرة الي كندا" فتأتي النتيجة بعشرات الالاف , وقد يكون فيها معلومات قديمة او اخبار ليست مفيدة للباحث , و هنا تأتي ميزة البحث ذو الدرجات



# Ranked retrieval

---

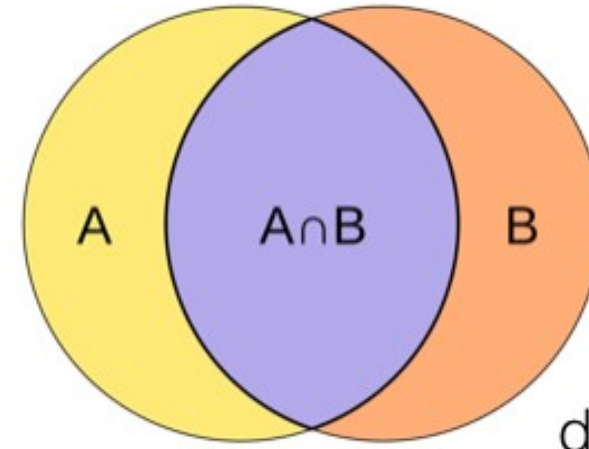
- Thus far, our queries have all been Boolean.
  - Documents either match or don't.
- Good for expert users with precise understanding of their needs and the collection.
  - Also good for applications: Applications can easily consume 1000s of results.
- Not good for the majority of users.
  - Most users incapable of writing Boolean queries (or they are, but they think it's too much work).
  - Most users don't want to wade through 1000s of results.
    - This is particularly true of web search.



إذن كيف يمكن عمل Scoring , هناك عدة طرق منها ما يسمى : معامل جاكارد Jaccard

و الفكرة الإحصائية لمعامل جاكارد قائمة علي أن نسبة التوافق بين القيمتين  $A$  ,  $B$  هي حاصل قسمة تقاطعهم علي اتحادهم , فلو لدينا دائرتين بمساحة معينة لكل منهما , فنسبة التوافق بينهما , هي مقدار التقاطع بينهما , مقسومة علي مساحة اتحادهما , و دائما ما يكون هذا الرقم يتراوح بين الصفر و الواحد

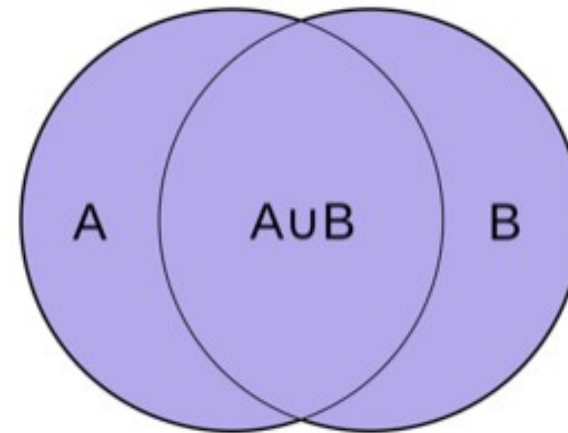
The intersect of A & B



division

$J(A,B) =$

The union of A & B



## Take 1: Jaccard coefficient

---

- A commonly used measure of overlap of two sets  $A$  and  $B$  is the Jaccard coefficient
- $\text{jaccard}(A,B) = |A \cap B| / |A \cup B|$
- $\text{jaccard}(A,A) = 1$
- $\text{jaccard}(A,B) = 0$  if  $A \cap B = 0$
- $A$  and  $B$  don't have to be the same size.
- Always assigns a number between 0 and 1.



فلو كان لدينا كلمات البحث هي جملة من 3 كلمات , و كان لدينا ملفين , كل منهما فيه عدد من الكلمات

فسيكون معامل جاكارد هو مقدار الكلمات المتقاطعة , مقسوم علي العدد الكلي للكلمات بين كلمات البحث و الملف ( مع حذف التكرار )

فلو كان جملتين هما :

- Spain is a beautiful country for tourism
- Picasso from Spain

و كان لدينا جملة البحث هي :

- Travel to Spain

فيكون معامل جاكارد للجملة الاولى هي  $7/1$  و الثاني  $3/1$

ولاحظ ان هذا غير دقيق , لان دقة الثاني اكبر من الاول , ليس لانه يقترب منه , لكن فقط لان عدد كلمات الجملة الثانية اقل

ولاحظ ايضا ان معامل جاكارد لا يهتم بعدد مرات تكرار الكلمات , وهذا شئ سلبي , لذا تم تعديل في القانون بعمل جذر للمقام لتقليل الاعتماد عليه

# Issues with Jaccard for scoring

- It doesn't consider *term frequency* (how many times a term occurs in a document)
  - Rare terms in a collection are more informative than frequent terms
  - Jaccard doesn't consider this information
- We need a more sophisticated way of normalizing for length
  - Later in this lecture, we'll use  $|A \cap B| / \sqrt{|A \cup B|}$  ... instead of  $|A \cap B| / |A \cup B|$  (Jaccard) for length normalization.

\* \* \* \* \*

نتحدث الآن عن تقييم نتائج محركات البحث , حيث يمكن تقييم نتائج محرك البحث عبر عدد من العوامل , مثل :

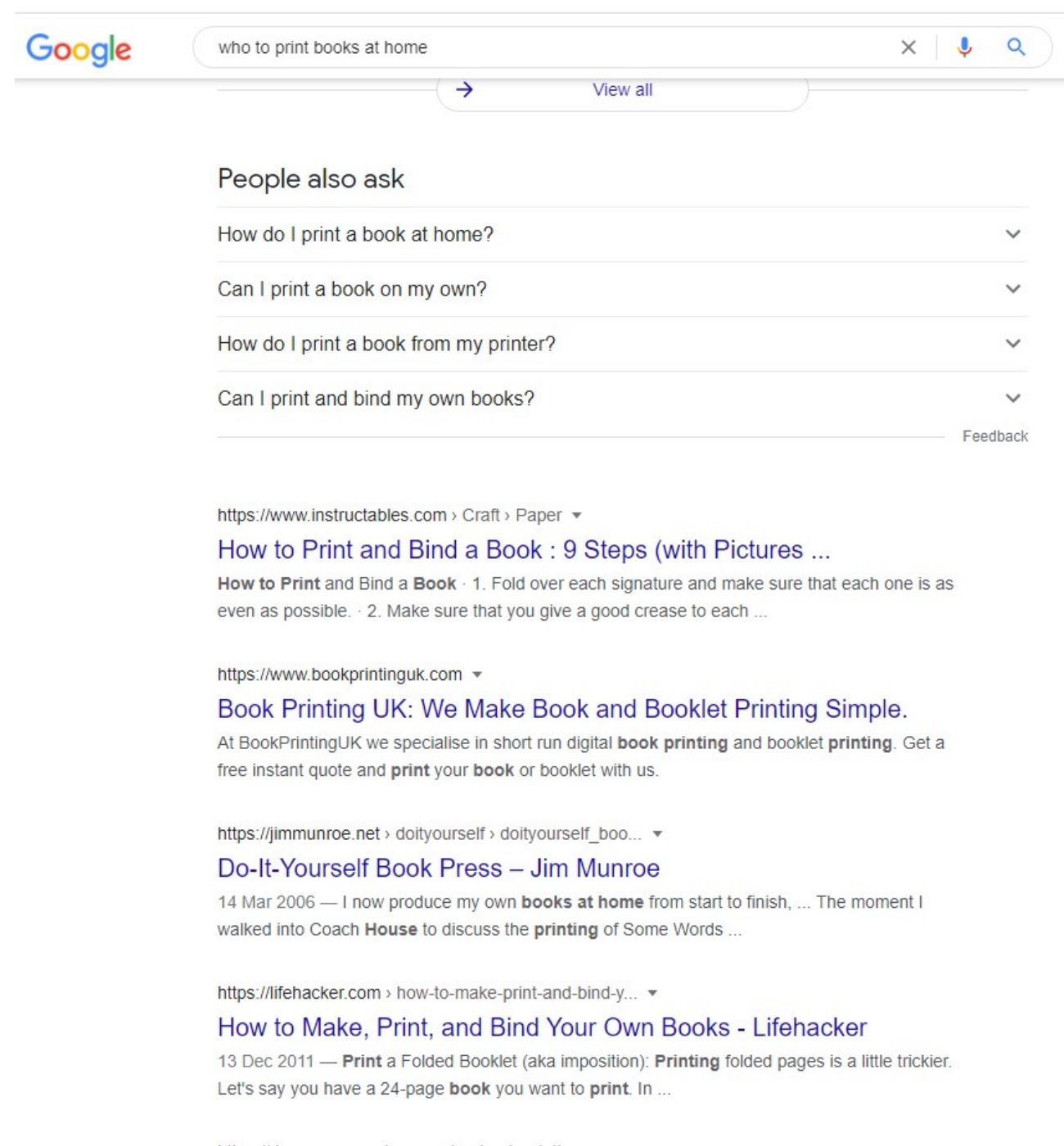
- سرعة تعامله مع الملفات
- عدد الملفات التي يبحث فيها
- سرعة البحث
- قدرته علي عرض معلومات معقدة

و بشكل ما , فالأهم من عوامل السرعة و الكمية , هي مقياس رضا المستخدم علي النتائج , و رضا المستخدم معتمدة علي علي مدي علاقة النتائج التي ستظهر له , بالشئ الذي يبحث عنه بالفعل , و بالتالي العامل الأهم للتناول هنا هو : مدي علاقة النتائج بكلمات البحث

فعلينا ان نعرف الفارق بين المعلومة المطلوبة , وكلمات البحث , فالمعلومة المطلوبة هي في ذهن المستخدم , بينما كلمات البحث هي ما يتعامل به مع المحرك , ليصل الي ما يريد

فلو كان لدينا جملة ما : الفارق بين السيارات اليدوية و الاوتوماتيك في استهلاك البنزين في درجات الحرارة العالية و تم استخدام كلمات هي : استهلاك البنزين يدوي اوتوماتيك حرارة

فهل سيقوم المحرك بإيجاد النتائج المطلوبة , ام ستكون النتائج متعلقة بالكلمات لكن بعيدة عن سؤال المستخدم





و تقوم محركات البحث بالتطوير الدائم , والتدريب علي عمليات البحث الادق , و هذا عبر معرفة اي رابط قام المستخدم بالضغط عليه لدي ادخاله كلمات بحث معينة , وبالتالي معرفة النتيجة الصحيحة

و يمكن قياس عملية البحث بناء علي :

- اذا كان المحرك لدي امكانية الدخول علي كمية كبيرة من الملفات
- اذا كانت كلمات البحث كافية
- اذا تم الربط بين كلمات البحث و النتيجة المطلوبة

لذا فإن عملية التقييم تتم عبر حساب : precision , recall , f1 score , وذلك عبر حصر عدد من نتائج البحث الأولي : و ليكن اول 10 نتائج , ثم فحصها يدويا , بمعرفة مدي علاقتها مع الشئ المطلوب , و حساب TP , TN , FP , FN

و يتم حساب ال precision لكل النتائج الاولى , كل علي حدة هكذا :

| Result No | Relevant | Precision |
|-----------|----------|-----------|
| 1         | Y        | 1.0       |
| 2         | N        | 0.5       |
| 3         | N        | 0.3       |
| 4         | Y        | 0.5       |
| 5         | Y        | 0.6       |
| 6         | N        | 0.5       |
| 7         | Y        | 4/7       |
| 8         | N        | 0.5       |
| 9         | Y        | 5/9       |
| 10        | Y        | 0.67      |

\* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \* \_ \*