Web Search using IR

Manuel Montoya Catalá

Multimedia Information Management

Carlos III University of Madrid

May, 2015

Introduction

Web Crawling + HTML processing

IR System

Experiments and Evaluation

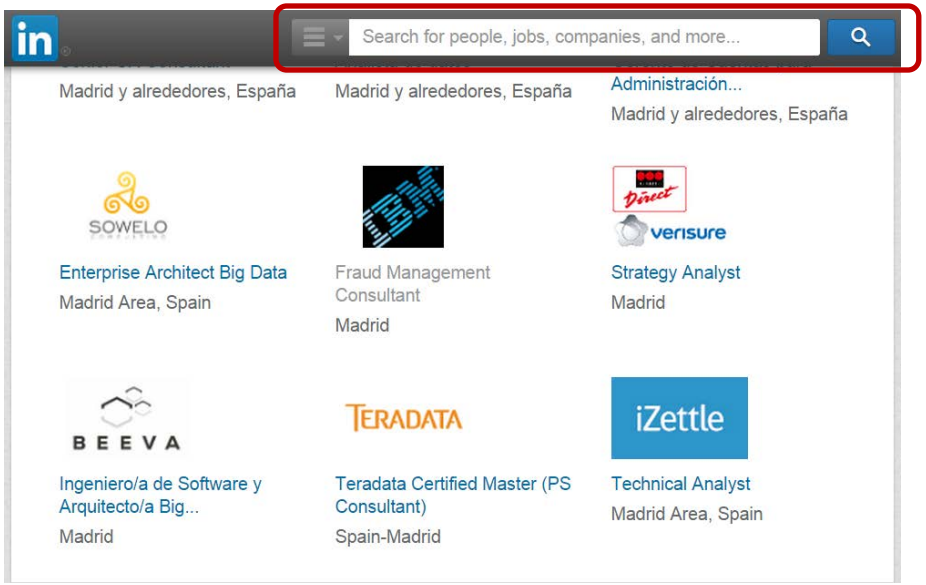
Conclusion

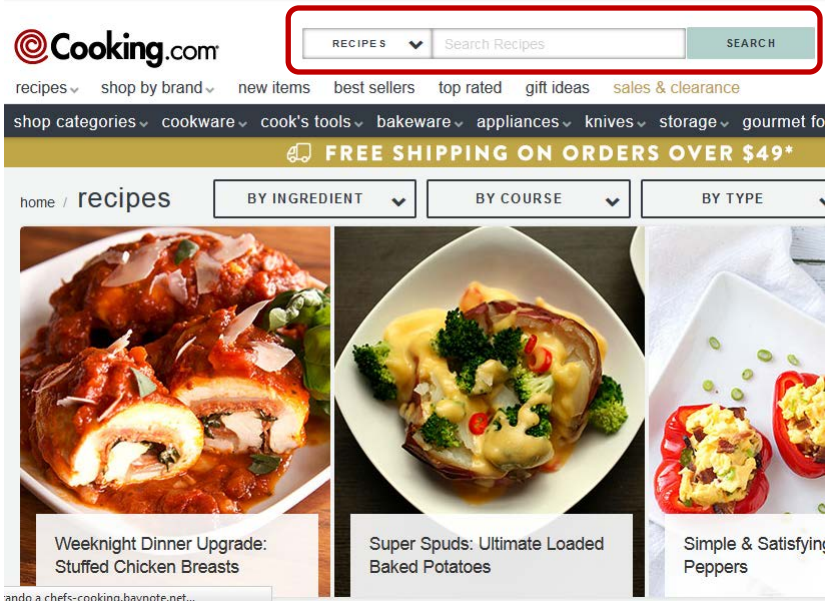
References

Introduction

The Internet

- 4,7 billion pages !!
- Lots of Multimedia Data





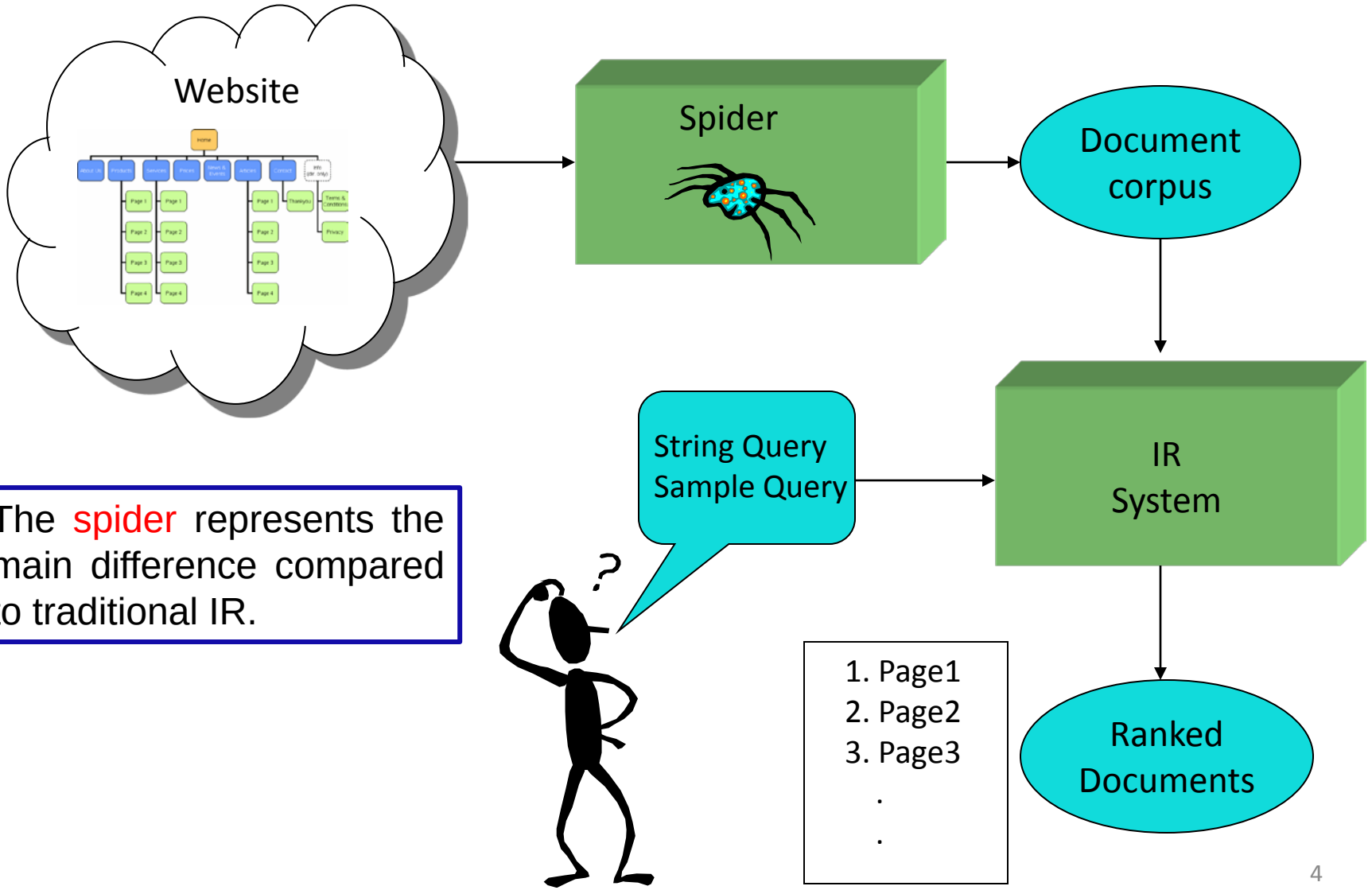
Too many documents for performing classical Information Retrieval

- Indexing and Page Rank
- Specific web searchers just constraint the URL.

Constraint the document number

- Specific topics
- Perform better IR models
- Allow Example Query

Structure



The spider represents the main difference compared to traditional IR.

Web Crawling + HTML processing

How do we download the Web pages ?

How do we obtain the:

- **Relevant text**
- **Metadata**

from the HTML documents?

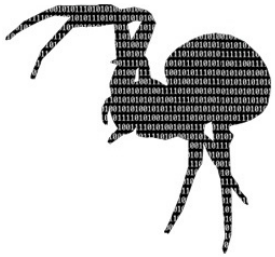
Web Crawling

We must download our corpus from the Internet using a Spider or WebCrawler.

A **Spider is a bot** that:

- Travels through WebPages downloading and procesing their content.

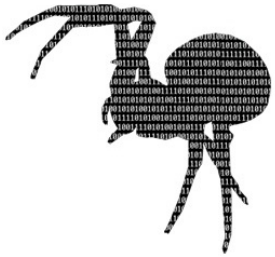
Parameters:



- Starting URLs
- Allowed domains
- Link-following protocol
- Rules for scraping.
- Document Processing function.

Tried to code a Spider in Python !! Worked fair enough !

- Not good enough for large scale downloads.
- Ended up using the **Scrapy framework.**



Web Crawling

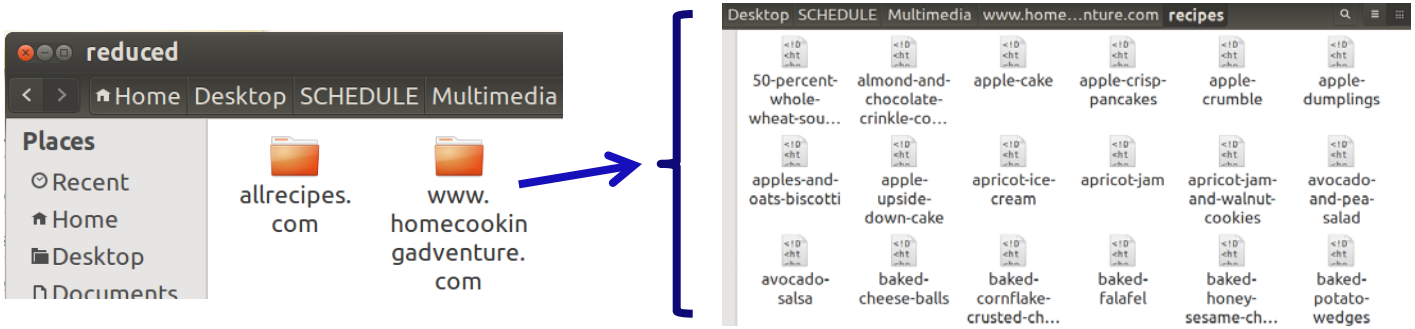
Scrapy is a scraping and web crawling framework, used to crawl websites and extract structured data from their pages.

```
111 class all_recipes_spider(CrawlSpider):
112     name = 'all_recipes'
113     allowed_domains = ["allrecipes.com"]  # Allowed domains
114     start_urls = ['http://allrecipes.com'] # urls from which the spider will start crawling
115     rules = [Rule(SgmlLinkExtractor(allow=[r'.*(recipe).*']), # Stracts all pages in the folder recepi
116          callback='parse_doc', follow=True),
117          ]
118     #.* means any number of chars
119     rules = [Rule(SgmlLinkExtractor(allow=[r'']), # Stracts all pages in the folder recepi
120           follow=True),  # To follow more shit
121          ]
122     def parse_doc(self, response):
123         filename = response.url # Writes a file with the name of the document
124         filename = filename[7:] # Eliminate the http:// part
125         if (filename[-1] == '/'):    # If the end of the URL is just a folder name
126             filename = filename + "index"  # Give a name to the main file
127         if not os.path.exists(os.path.dirname(filename)): # If the folder of this does not exist
128             os.makedirs(os.path.dirname(filename))        # Make the folder
129         with open(filename + ".html", 'wb') as f:
130             f.write(response.body)
```

URL properties

Crawling Rules

Processing



Corpus Preprocessing

Once we have folders structures containing **HTML documents** we have to:

- Read all possible documents, need a «walker» program.
- Remove non-recipe documents.
- Remove near duplicates.
  - Due to JavaScript generating pages

An HTML document has:

- Plain text.
- HTML markups
- Scripts: JavaScript, PHP…
- Syles: CSS
- Hyperlinks

```
<ul id="ulNutrient" class="nutrSumList">
    <li class="categories">Sodium</li>
    <li class="units"><span id="lblNutrientValue">434
    <li class="nutrition-rating">
        <ul>
            <li id="divNutrientGradient" class="nutri
            <li class="nutrition-rating-img"></li>
        </ul>
    </li>
    <li class="percentages">17%</li>
</ul>

<p>* Percent Daily Values are based on a 2,000 calorie diet.<
<a href="javascript:void(0);" id="nutritionSeeMore" class="rp
<div id="eshaLinkSummary"><a id="eshaLink" rel="nofollow" tar

</div>

<div id="nutritionDetail" style="display:none;" class="bottomLaye

<!-- DETAILED NUTRITION -->
<div id="nutritiontable" class="nutrDetWrap" itemprop="nutrit

    <h2>Nutritional Information</h2>
    <p class="recipeTitle"><span id="lblTitle">Amazing Slow C
    <p class="top">
        Serving Size: <span itemprop="servingSize">1/12 of a
        Servings Per Recipe: 12<br />
        <strong>Amount Per Serving</strong><br />
        Calories: <strong><span id="litCalories" itemprop="ca
        Calories from Fat: <strong>102</strong></p>
    <ul class="nutrDetList">
        <li class="medBorder">
            <span class="right">% Daily Value *</span>
        </li>
        <li>
            <span class="left"><span></span>Total Fat</span>
```

We need to **extract the relevant information**:

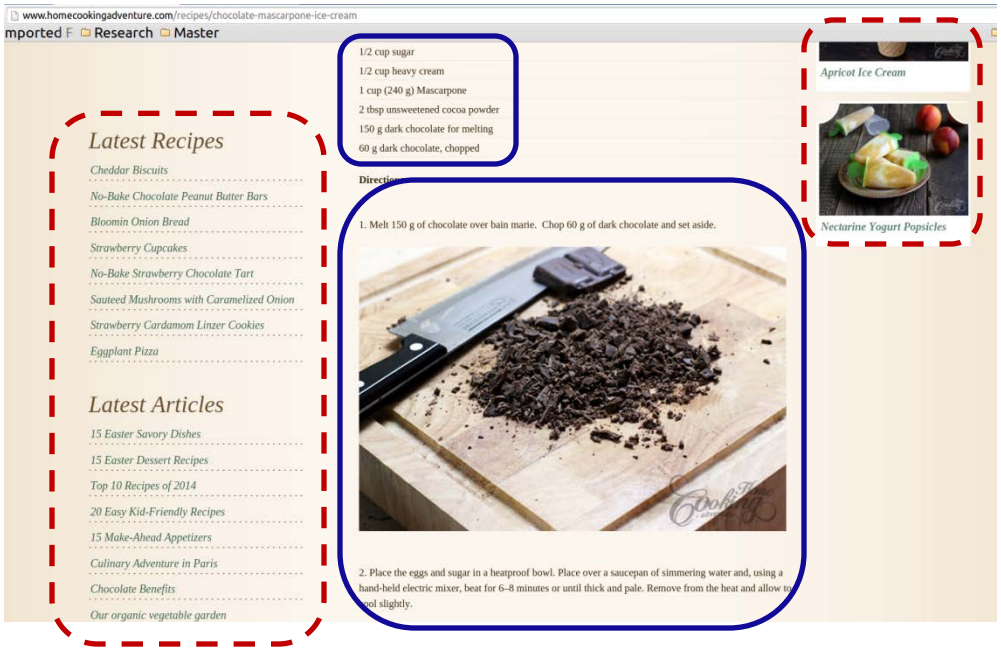
- Remove Scripts and Styles
- Remove the markups.
- Get Relevant text

Beautiful Soup !!

Corpus Preprocessing

BeautifulSoup library for HTML processing

- Remove Scripts and Styles
- Reference parts of the HTML
- Get Plain texts
- **We still have to get our hands into the HTML code !!**




```
        <div id="ingredients_bg">
          <ul><li><span class="ingheading">Makes about 4 servir
        </div>
      </div>
      <div id="preparation">
      <div class="title" >Directions</div>
      <div itemprop="instructions">
      <ol>
<li><span>
    Preheat oven to 400F (200C) and line a baking sheet with parch
<li><span>
    Beat the eggs with oil and set aside. Place the flour on a pla
<li><span>
```

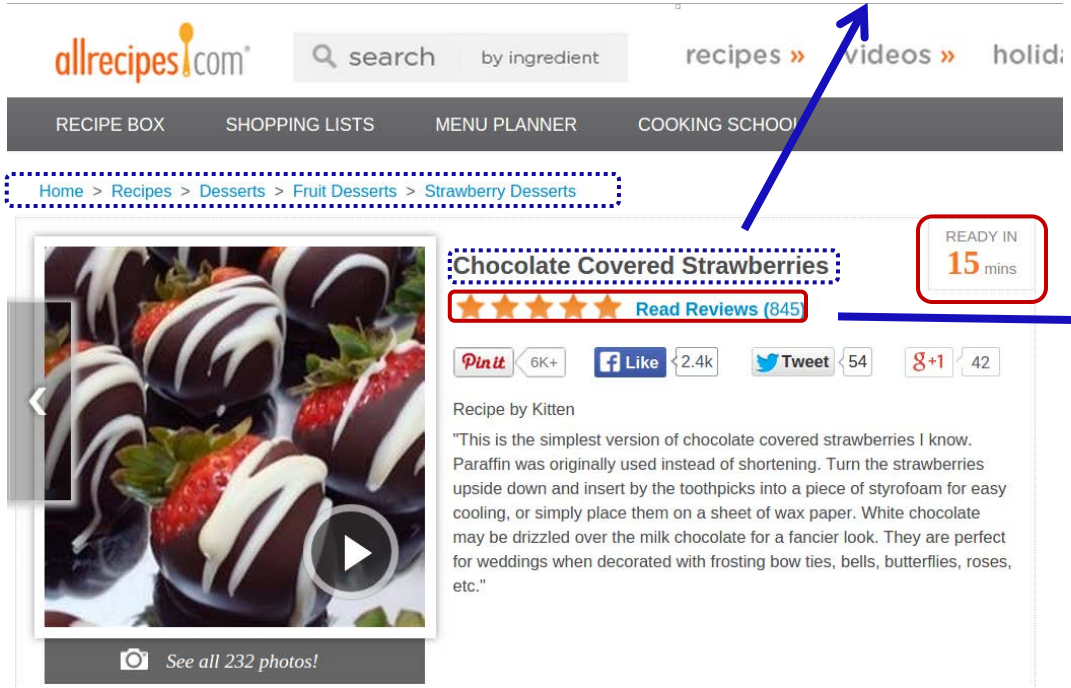
```
      <span itemprop="servingSize"><b>1 Chicken Strip</b></span> - Cal
</div>
        <div style="height: 7px; font-size: 7px; margin-bottom:10px;

<div id="posted_comments" style="margin-top:20px;">
                          <div class="comment_top">
    On February 03, 2014 at  08:19 pm, <span><a href="http://cookingact
</div>
<div class="comment_middle">
    I LOOOVE how crisp and delicious these look!! And, of course, that
</div>
<div class="comment_bottom">
</div>
```

Corpus Preprocessing

Get **Metadata** Information for Ranking:

- **KeyWords**
- **Ratings**

```
<div id="recipe_title" ><h1 itemprop="name">Baked Cornflake Crus
<meta itemprop="author" content="Home Cooking Adventures" />
<div id="main_img">
            <img src="/images/recipes/baked-cornflake-crusted-ch
<div style="min-height:390px; margin-top:10px;">
```




```
▼<div class="rating-stars-
img" title="Most cooks
definitely will make this
recipe again">
    <meta itemprop=
    "ratingValue" content=
    "4.80523">
  </div>
</div>
```

IR system

How do we preprocess the Plain text ?

Represent documents as an TF-IDF vector?

What Similarity Measures do you use ?

Text Preprocessing

Once we have relevant plain text, we use the **NLTK** Python library for preprocessing

- Tokenization
- Lower Case
- Remove non-alphanumeric
- Remove stopwords
- Steamming (Snowball)

```
def doc_tokeniz(document):
    # Gets the tokens of the document. The tokens are words.
    tokens = word_tokenize(document)
    return tokens

def doc_lowercase (document):
    # Transforms a list of words (document) into lowercase
    low_text = [w.lower() for w in document]
    return low_text

def doc_rem_stopwords(document):
    # Removes stopwords obtained from the nltk english corpus.
    stopwords_en = stopwords.words('english')
    clean_text = [word for word in document if not word in stopwords_en]
    return clean_text

def doc_stem(document):
    # Performs the english steaming of the words
    stemmer = SnowballStemmer('english')
    steammed_text = [stemmer.stem(word)for word in document]
    return steammed_text

def doc_rem_punctuation(document):
    # Removes punctuation marks
    clean_text = [w for w in document if w.isalnum()]
    return clean_text
```

String Query !

```
I'm feeling like CHocolate with strawberry chickeN salad Flavour. I love chicken !

['I', "'m", 'feeling', 'like', 'CHocolate', 'with', 'strawberry', 'chickeN', 'salad', 'Flavour', '.', 'I', 'love', 'chicken', '!']

['i', "'m", 'feeling', 'like', 'chocolate', 'with', 'strawberry', 'chicken', 'salad', 'flavour', '.', 'i', 'love', 'chicken', '!']

['i', 'feeling', 'like', 'chocolate', 'with', 'strawberry', 'chicken', 'salad', 'flavour', 'i', 'love', 'chicken']

['feeling', 'like', 'chocolate', 'strawberry', 'chicken', 'salad', 'flavour', 'love', 'chicken']

[u'feel', u'like', u'chocol', u'strawberri', u'chicken', u'salad', u'flavour', u'love', u'chicken']
```

Vector Representation

We represent each HTML document as a vector of float numbers.

- We perform the TF-IDF of every HTML document
- $TF - IDF_{(t,d,c)} = TF_{(t,d)} \cdot IDF_{(t,c)}$
  - $TF_{(t,d)}$: Term Frecuency (BoW). Frecuency of term 't' in the document 'd'
  - $IDF_{(t,c)}$: Inverse Document Frecuency of term 't' in corpus 'c'

$$TF_{(t,d)} = \frac{Number\ of\ terms\ 't'\ in\ document\ 'd'}{Total\ number\ of\ terms\ in\ 'd'} \qquad IDF_{(t,c)} = \log_2\left(\frac{Total\ number\ of\ documents\ in\ 'c'}{Number\ of\ documents\ that\ contain\ 't'}\right)$$

String Query !

1.98562278646	chicken
1.54963647625	strawberri
0.946514999817	chocol
0.812983607193	like
1.08696619255	salad
1.6785510309	feel
0.775371896574	love
2.74858411265	flavour

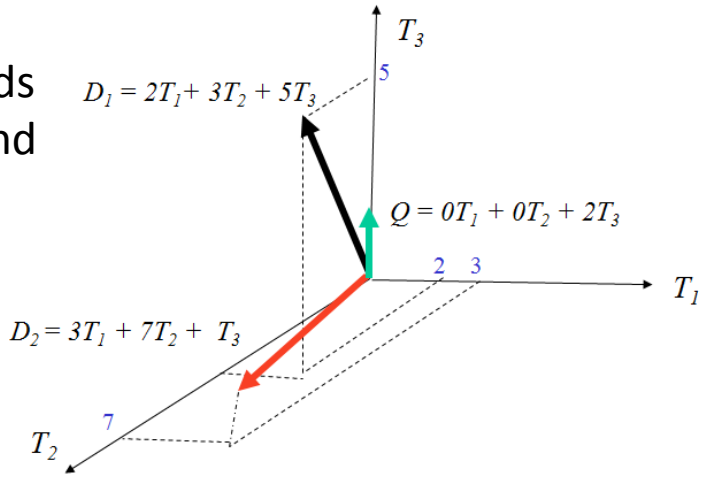
Similarity measures

Given a **plain text Query D**, we need to:

- Get its TF-IDF vector.
- Compare it with the whole corpus according to a Similarity Measure.

Programming Difficulty:

- Every document has a different set of words
- Need to find the words in commun and their associate TF-IDF values.



Similarity Measures:

- Euclidean Distance
- Cosine similarity

Good for Example Queries !

Bad for short String Queries and Keywords !

Simmilarity for short Queries

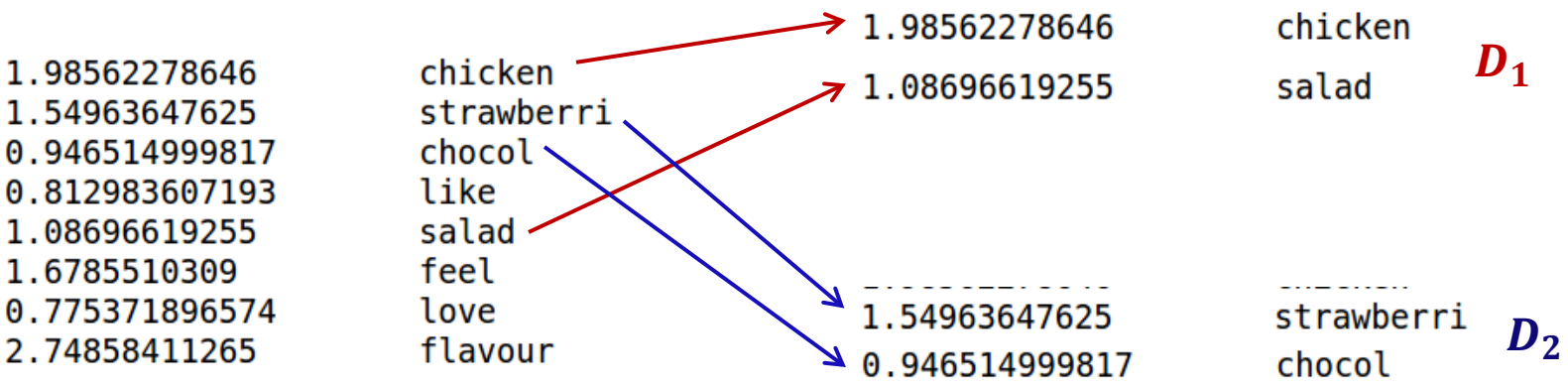
For short Queries, the comparasing between the Quert TF-IDF does not make sense

- Cannot stract real frecuency profile of query.
- Most values are zero and they do not repeat.

Idea !!

- Add up the TF-IDF values of the commun words with the query.

$$Sim(Q, D_d) = \sum_{t \in (Q \cap D_d)} TF - IDF_{(t,d)}$$



Ranking

The BEST similarity measure depends on the properties of the query:

- For **Example Queries**:
  - ✓ Average ranking of Ecuclidean Distance + Cosine Distance rankings
- For **String Queries:**
  - ✓ Sum of the TFIDF values of the common words.
- For **keywords**:
  - ✓ Number of words in commun.
- For **Stars Ranking**:
  - ✓ Rule-Based Constraint

Experiments and Results

What is your Corpus made of ?

Are your Results any Good?

What Similarity Measures do you use ?

Experiments

Corpus of about 1000 recipes !! Not too big to avoid very close recipes !

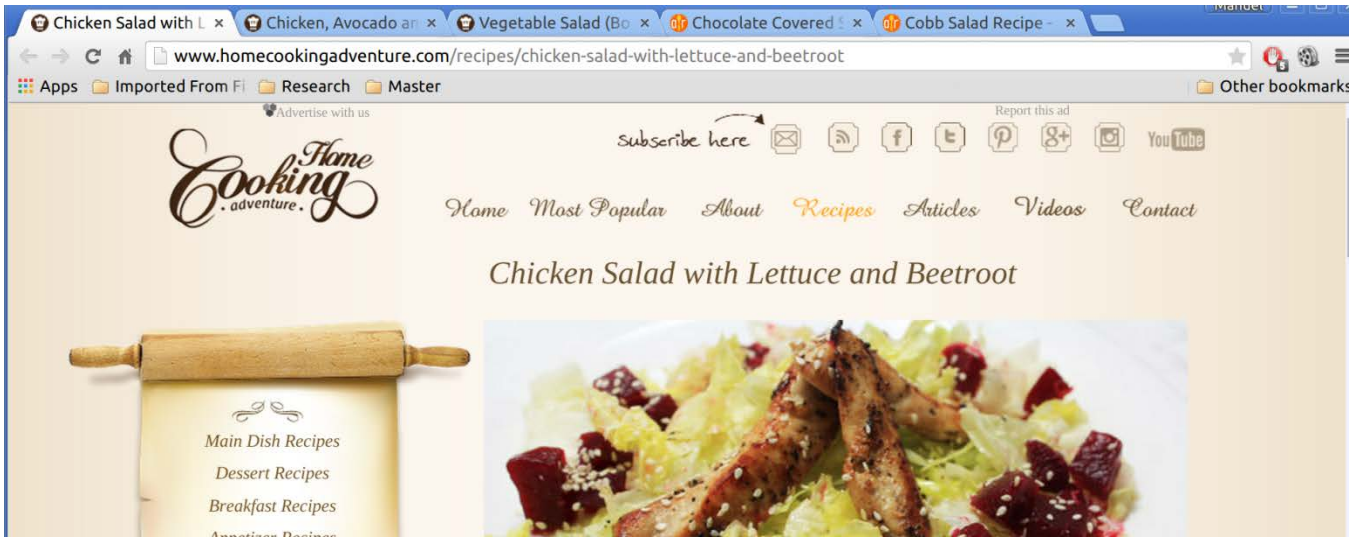
- www.homecookingadventure.com
- allrecipes.com

System accepts:

- Plain text queries
- URLs as a query. It downloads and preprocess the HTML document.

The System then opens up a browser with the top 5/10/20… recipes !!

- Using the «webbrowser» library

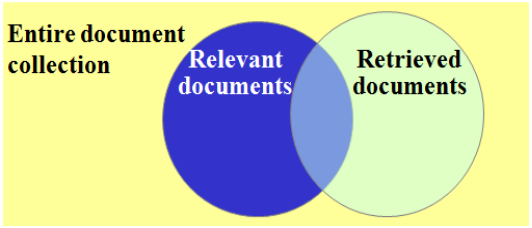


Demonstration Time !

Give me some ingredients !

Try to be nice !

Results and Evaluation



Results were evaluated using only **MAP** with $K = 5$

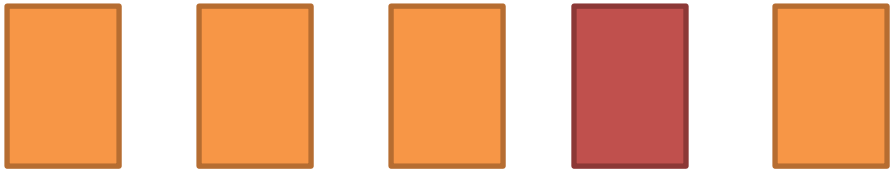
- Mean Average Precision
- Using **Explicit Relevance Feedback**

Our corpus is dynamic and unlabeled:

- It's costly to perform Recall or Cumulative Gain.

$$Precisiont\ at\ document\ t = \frac{Number\ of\ relevant\ pages\ retrieved}{Total\ number\ of\ pages\ retrieved}$$

On average they follow this structure:



$$Average\ Precision = \frac{1}{4}(1 + 1 + 1 + 0.8\ ) = 0.95$$

Conclusions

You can make your **own specific corpus** !!

- About anything you want !
- In one single day !
- For FREE !

The System:

- Retrives good HTML documents.
- Performs more accuarate Web searchs than Google jejej
- Constrains the search using specific inner web data
- Can use example queries.

References

[1] Brants, Thorsten. "Natural Language Processing in Information Retrieval." *CLIN*. 2003.

[2] WU, Ho Chung, et al. Interpreting tf-idf term weights as making relevance decisions. *ACM Transactions on Information Systems (TOIS)*, 2008, vol. 26, no 3, p. 13.

[3] D. Jurafsky, C. Manning. “Natural Language Processing”. By Standford. Coursera: https://class.coursera.org/nlp/lecture

[4] Broder, Andrei. "A taxonomy of web search." *ACM Sigir forum*. Vol. 36. No. 2. ACM, 2002.

[5] Mansourian, Yazdan. "Similarities and differences between web search procedure and searching in the pre-web information retrieval systems." *Webology* 1.1 (2004).

[6] Langville, Amy N., and Carl D. Meyer. "Information retrieval and web search." *Supported by National Science Foundation under NSF grant CCR-0318575, USA* (2006).

Thank you !!