İSTANBUL ŞEHİR UNIVERSITY

ENGR 212: Programming Practice

Week 11

İSTANBUL ŞEHİR UNIVERSITY

Searching and Ranking

Ranking measures

İSTANBUL ŞEHİR UNIVERSITY

- **Word frequency:** the number of times the words in the query appear in the document can help determine how relevant the document is.
- **Document location:** the main subject of a document will probably appear near the beginning of the document.
- **Word distance:** if there are multiple words in the query, they should appear close together in the document.

Document Location

İSTANBUL ŞEHİR UNIVERSITY

- Search term's location in the page.
- If a page is relevant to the search term, it will appear closer to the top of the page, perhaps even in the title.
- Score results higher if the query term appears early in the document.
- wordlocation **table: the locations of the words were recorded.**

Document Location

İSTANBUL ŞEHİR UNIVERSITY

```
def locationscore(self, results):
    locations=dict([(url, 1000000) for url in results])
    for url in results:
        score = 0
        for wordlocations in results[url]:
            score += min(wordlocations)
        locations[url] = score
    return self.normalizescores(locations, smallIsBetter=True)
```

Document Location

İSTANBUL ŞEHİR UNIVERSITY

- change the weights line in getscoredlist to this:

weights=[(1.0, self.locationscore(rows))]

```
import mysearchengine
dbtables = {'urllist': 'urllist.db', 'wordlocation':'wordlocation.db',
      'link':'link.db', 'linkwords':'linkwords.db', 'pagerank':'pagerank.db'}
searcher = mysearchengine.searcher(dbtables)
searcher.query('computer science')
```

Word Distance

İSTANBUL ŞEHİR UNIVERSITY

- When a query contains multiple words, seek results in which the words in the query are close to each other in the page.
- Multiple-word queries, seek pages that conceptually relates the different words.

Word Distance

İSTANBUL ŞEHİR UNIVERSITY

Left as a take-home exercise!

Link information

İSTANBUL ŞEHİR UNIVERSITY

- Used scoring metrics based on the content of the page.
- How about information that others have provided about the page, who linked to the page, what they said about it?

Pages created by spammers are less likely to be linked than pages with real content.

Link information

İSTANBUL ŞEHİR UNIVERSITY

- The link table has the URLs for the source and target of every link that it has encountered.
- The linkwords table connects the words with the links.

Simple Count

İSTANBUL ŞEHİR UNIVERSITY

- Count inbound links on each page and use the total number of links as a metric for the page.
- The scoring function based on this count:

```
def inboundlinkscore(self, results):
    inboundcount=dict([(url, len(self.link[url]))
                       for url in results if url in self.link])
    return self.normalizescores(inboundcount)
```

Simple Count

İSTANBUL ŞEHİR UNIVERSITY

- Using this metric by itself will simply return all the pages containing the search terms,

  "*ranked solely on how many inbound links they have.*"

- We need to combine the inbound-links metric with one of the content/relevance metrics we saw earlier.

What if?

İSTANBUL ŞEHİR UNIVERSITY

- Someone can easily set up several sites pointing to a page whose score they want to increase.

What about!

İSTANBUL ŞEHİR UNIVERSITY

- What about results that have attracted the attention of very **popular** sites?
- How to make *links from popular pages worth more* in calculating rankings?

İSTANBUL ŞEHİR UNIVERSITY

PageRank

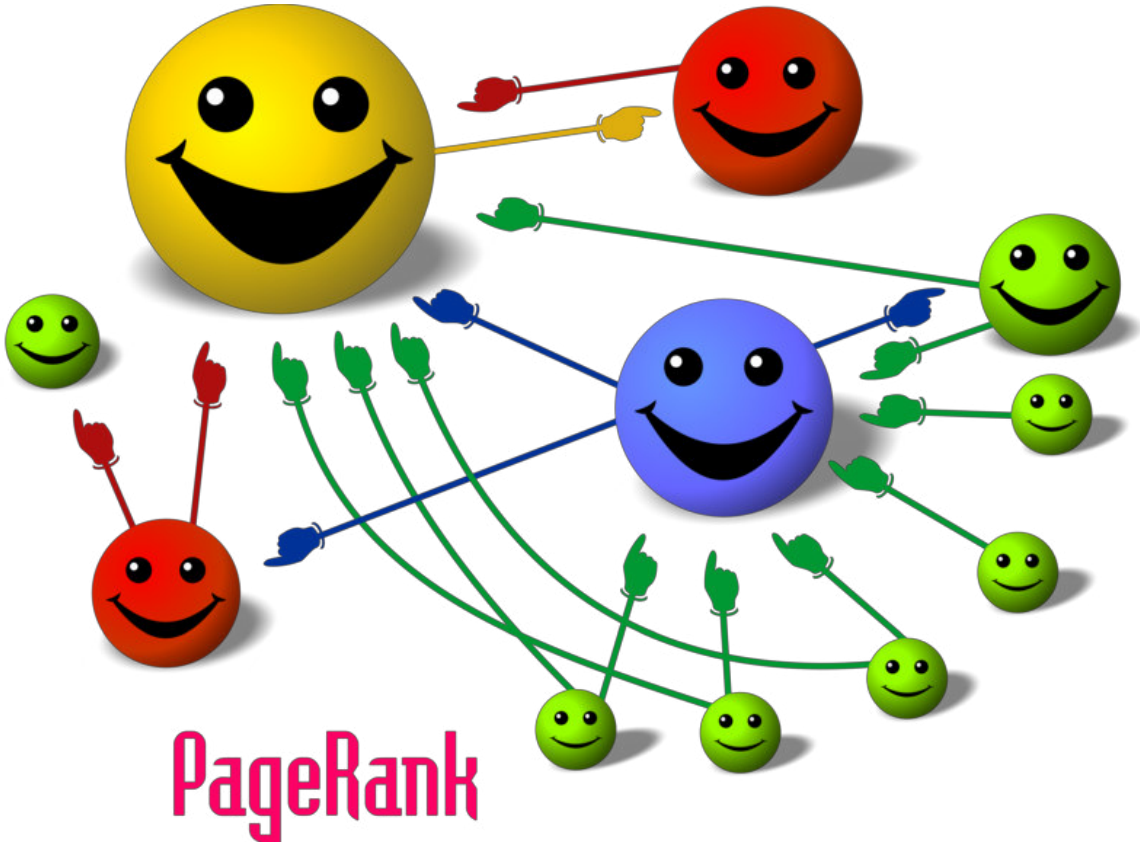
PageRank

İSTANBUL ŞEHİR UNIVERSITY

- **Page**Rank (named after Larry **Page**) calculates the probability that someone randomly clicking on links will arrive at a certain page.
- The more inbound links the page has from other popular pages, the more likely it is that someone will end up there purely by chance.

PageRank

- Basic principle of PageRank.
  - The size of each face is proportional to the total size of the other faces which are pointing to it.



Source: http://en.wikipedia.org/wiki/PageRank

PageRank

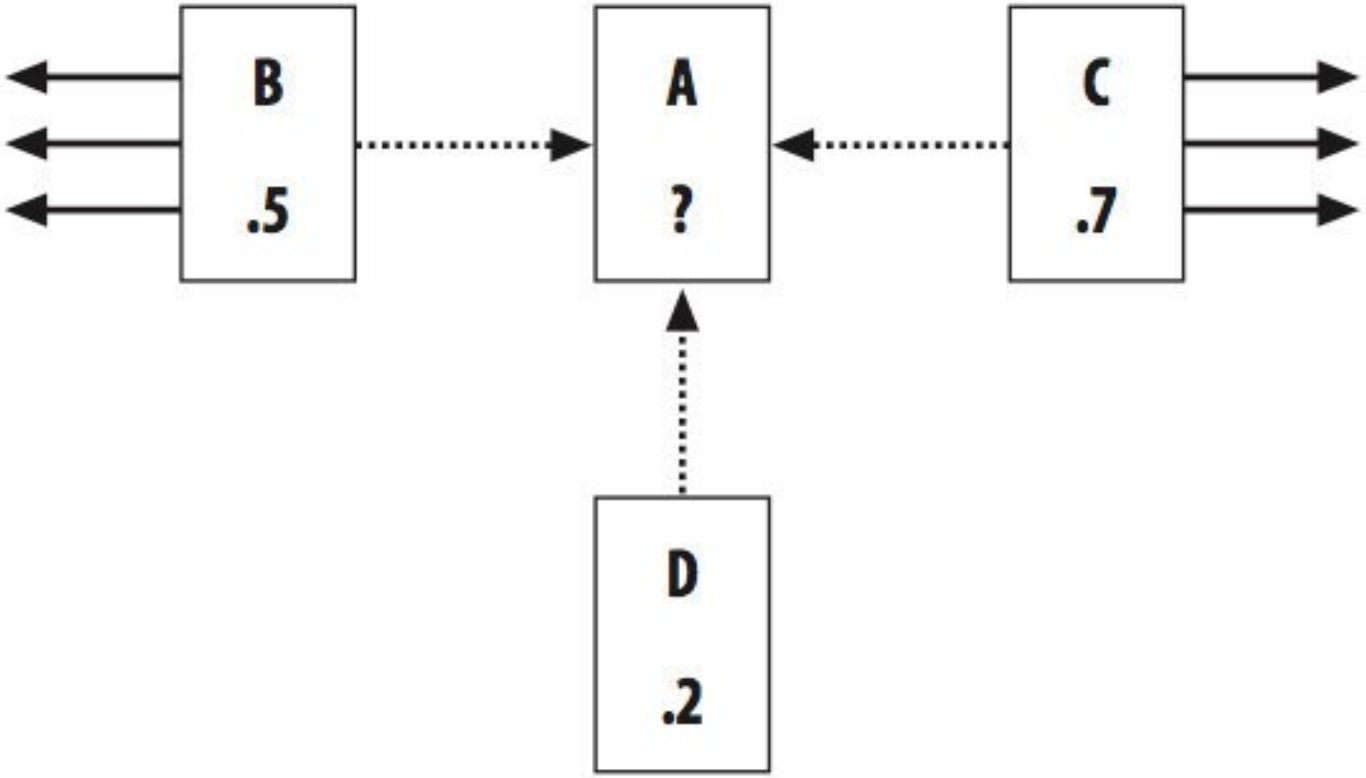
İSTANBUL ŞEHİR UNIVERSITY

- If the user keeps clicking forever, they'll eventually reach every page, but most people stop surfing after a while.
- To capture this, PageRank also uses a damping factor of 0.85, indicating that

  there is an 85 percent chance that a user will continue clicking on links at each page.

PageRank

İSTANBUL ŞEHİR UNIVERSITY



PageRank

İSTANBUL ŞEHİR UNIVERSITY

- PR(A) = 0.15 + 0.85 *
  (PR(B)/links(B) + PR(C)/links(C) + PR(D)/links(D))

  = 0.15 + 0.85 * (0.5/4 + 0.7/5 + 0.2/1)
  = 0.15 + 0.85 * (0.125 + 0.14 + 0.2)
  = 0.15 + 0.85 * 0.465
  = 0.54525

There is a small catch!

İSTANBUL ŞEHİR UNIVERSITY

- All the pages linking to A already had PageRanks.
- You can't calculate a page's score until you know the scores of all the pages that link there, and you can't calculate their scores without doing the same for all the pages that link to them.
- How is it possible to calculate PageRanks for a whole set of pages that don't already have PageRanks?

İSTANBUL ŞEHİR UNIVERSITY

AND YET THE QUESTION REMAINED:
"WHO CAME FIRST?"

PageRank

İSTANBUL ŞEHİR UNIVERSITY

- The solution is to set all the PageRanks to an initial arbitrary value (*the code will use 1.0, but the actual value doesn't make any difference*).
- Repeat the calculation over several iterations.
- After each iteration, the PageRank for each page gets closer to its true PageRank value.
- The number of iterations needed varies with the number of pages.

PageRank computation


```
def calculatepagerank(self,iterations=20):
    self.pagerank = shelve.open(self.dbtables['pagerank'],
                                writeback=True, flag='n')

    # initialize every url with a page rank of 1
    for url in self.urllist:
        self.pagerank[url] = 1.0

    for i in range(iterations):
        print "Iteration %d" % (i)
        for url in self.urllist:
            pr=0.15
            # Loop through all the pages that link to this one
            if url in self.link:
              for linker in self.link[url]:
                  linkingpr = self.pagerank[linker]

                  # Get the total number of links from the linker
                  linkingcount = self.urllist[linker]
                  pr += 0.85*(linkingpr/linkingcount)

            self.pagerank[url] = pr
```

PageRank computation


```
>>> reload(mysearchengine)
>>> crawler=searchengine.crawler(dbtables)
>>> crawler.calculatepagerank()
```

Score Combination

İSTANBUL ŞEHİR UNIVERSITY

- weights = [(1.0, self.locationscore(rows)),
  (1.0, self.frequencyscore(rows)),
  (1.0, self.pagerankscore(rows))]

İSTANBUL ŞEHİR UNIVERSITY

Document Filtering

Document Filtering

İSTANBUL ŞEHİR UNIVERSITY

- Your email address in the wrong hands:

  lots of unnecessary and unsolicited email messages!!!

- ***SPAM SPAM SPAM SPAM SPAM SPAM SPAM SPAM SPAM SPAM SPAM SPAM ...***

- A well-known application of document filtering is the elimination of spam.

Classification

İSTANBUL ŞEHİR UNIVERSITY

- The algorithms are not specific to dealing with spam.
- The general problem of learning to recognize whether a document belongs in one category or another.
- ***App 1*** : Automatically dividing your inbox into social and work-related email, based on the contents of the messages.
- ***App2***: Identifying email messages that request information and automatically forwarding them to the most competent person to answer them.

Filtering Spam

İSTANBUL ŞEHİR UNIVERSITY

- Early attempts to filter spam: rule-based classifiers.
  - e.g., overuse of capital letters, ...
- Spammers learned all the rules and stopped exhibiting the obvious behaviors to get around the filters.
- "What is spam"? Subjective!
- How about tailor-fitting?
  You teach me what is spam email and what isn't spam email, and I learn for you to automate the task.

Documents and Words

- The classifier needs features to use for classifying different items.
- Some words are more likely to appear in spam than in not-spam?
- Use words in the document as features.
- Not just individual words, however; word pairs or phrases or anything else that can be classified as absent or present in a particular document.

Determining features

İSTANBUL ŞEHİR UNIVERSITY

- The entire text of the document could be a feature?
- At the other extreme, the features could be individual characters?
- The choice to use **words** of an email message as features:
  - how to divide words?
  - which punctuation to include?
  - whether header information should be included?

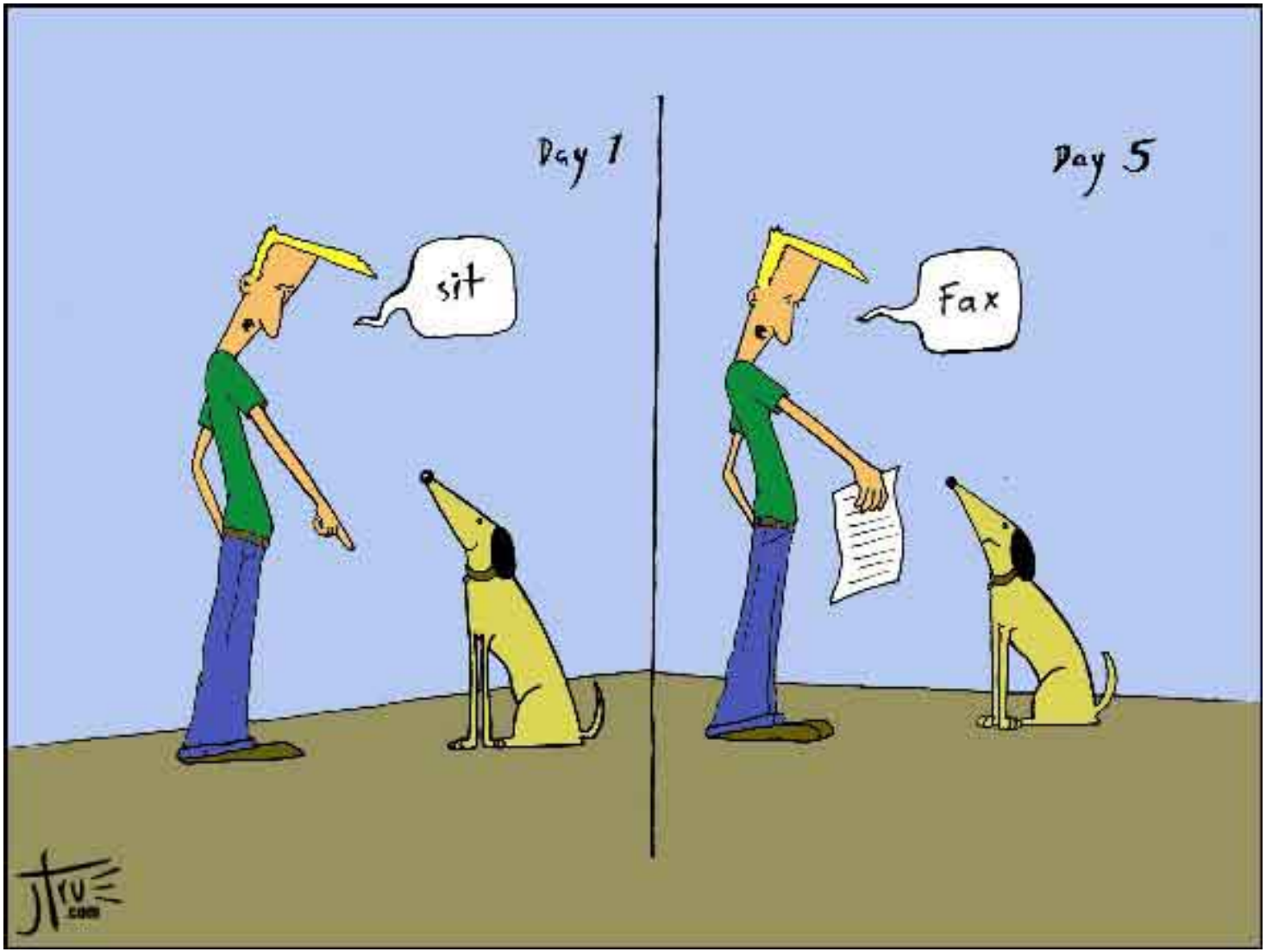
Extracting Features from Text

İSTANBUL ŞEHİR UNIVERSITY

```
import re
import math

def getwords(doc):
  splitter=re.compile(r'\W*')

  # Split the words by non-alpha characters
  words=[s.lower( ) for s in splitter.split(doc)
         if len(s)>2 and len(s)<20]

  # Return the unique set of words only
  return dict([(w,1) for w in words])
```

Training the Classifier

İSTANBUL ŞEHİR UNIVERSITY



Training the Classifier

İSTANBUL ŞEHİR UNIVERSITY

- The more examples the classifier is fed with, the better the classifier will get at making predictions.
  - example: a document and its classification
- The classifier starts off very uncertain and increase in certainty as it "learns"
  - which features are important for making a distinction.

Creating classifier

İSTANBUL ŞEHİR UNIVERSITY

```
class classifier:
  def __init_ _(self,getfeatures,filename=None):
    # Counts of feature/category combinations
    self.fc={}

    # Counts of documents in each category
    self.cc={}
    self.getfeatures=getfeatures
```

Training the Classifier

İSTANBUL ŞEHİR UNIVERSITY

fc:

```
{
    'python': {'bad': 0, 'good': 6},
    'the':    {'bad': 3, 'good': 3}
}
```

cc:

```
{
    'good': 45,
    'bad':  68
}
```

Creating classifier

İSTANBUL ŞEHİR UNIVERSITY

```
# Increase the count of a feature/category pair
def incf(self,f,cat):
   self.fc.setdefault(f,{})
   self.fc[f].setdefault(cat,0)
   self.fc[f][cat]+=1

# Increase the count of a category
def incc(self,cat):
   self.cc.setdefault(cat,0)
   self.cc[cat]+=1

# The number of times a feature has appeared in a category
def fcount(self,f,cat):
   if f in self.fc and cat in self.fc[f]:
      return float(self.fc[f][cat])
   return 0.0
```

Creating classifier

İSTANBUL ŞEHİR UNIVERSITY

```
# The number of items in a category
def catcount(self,cat):
   if cat in self.cc:
      return float(self.cc[cat])
   return 0

# The total number of items
def totalcount(self):
   return sum(self.cc.values( ))

# The list of all categories
def categories(self):
   return self.cc.keys( )
```

Creating a Classifier – Train method

İSTANBUL ŞEHİR UNIVERSITY

```
def train(self,item,cat):
    features=self.getfeatures(item)
    # Increment the count for every feature
    # in this category
    for f in features:
        self.incf(f,cat)

    # Increment the count for this category
    self.incc(cat)
```

Let's check if our classifier works correctly so far!

İSTANBUL ŞEHİR UNIVERSITY

```
>>> import docclass
>>> cl = docclass.classifier(docclass.getwords)
>>> cl.train('the quick brown fox jumps over the lazy dog', 'good')
>>> cl.train('make quick money in the online casino', 'bad')
>>> print cl.fcount('quick','good')
>>> print cl.categories()
```

Creating a Classifier – Sample Train method


```
def sampletrain(cl):
    cl.train('Nobody owns the water.','good')
    cl.train('the quick rabbit jumps fences','good')
    cl.train('buy pharmaceuticals now','bad')
    cl.train('make quick money at the online  casino','bad')
    cl.train('the quick brown fox jumps','good')
```

Calculating probabilities

- We have counts for how often email messages appear in each category (after training).
- The probability that a word is in a particular category C:

$$= \frac{\text{\# of times ``word'' appears in a document in C}}{\text{the total number of documents in C}}$$

Calculating probabilities


```
def fprob(self,f,cat):
    if self.catcount(cat) == 0:
          return 0
    # The total number of times this feature appeared in
    # this category divided by the total number of items
    # in this category
    return self.fcount(f,cat)/self.catcount(cat)
```