

# HW Solr Overview

**There are several components to this homework; you will possibly not need all of them;**

- 1. Installing Ubuntu on Windows**
- 2. Installing Solr**
- 3. Using Solr to Index a web site**
- 4. the exercise - comparing ranking algorithms**
- 5. What to submit**
- 6. Grading guidelines**

# What you will need to do

## SUMMARY (1 of 2)

- 1. If you are using Windows, you will have to install Ubuntu on your machine**
  - if you have a Mac, skip to step 2
  - Solr requires a Unix platform
  - first you must download and install Oracle's VirtualBox, free software that will permit you to install Ubuntu on your Windows device
  - second you will download and install Ubuntu within Virtualbox
- 2. Download and install the current release of Solr**
  - there is a Quick Start Guide to help you at <http://lucene.apache.org/solr/quickstart.html>
  - the Solr server should be started; to check that it is running look at
  - <http://localhost:8983/solr>
- 3. Download the reference news website you are responsible for**
  - all news web pages are located on Google drive and are accessible using
  - <https://drive.google.com/drive/u/1/folders/0B1xPYK9J4gtKSTg4b0d2Qm1DLWM>

# What you will need to do

## SUMMARY (2 of 2)

4. **Import the news website web pages into Solr**
5. **Using your laptop install or use an existing web server to create a website**
  - Macs (I believe) have Apache pre-installed
  - in your website create a web page that looks like a Google query box
  - in your website write a program that takes a query and sends it to Solr
  - in your website write a program that accepts the results from Solr and displays them as a web page
6. **Run the set of 8 queries at the end of Homework4.pdf**
  - save the top ten results for each of the eight queries
7. **Use the NetworkX library, the downloaded web pages and Jsoup library to create a network graph of the downloaded web pages**
  - use the PageRank function from NetworkX to create a file, pagerankFile.txt that contains individual page ranks for every downloaded web page
  - install the pagerankFile in Solr and direct Solr to use it on queries
8. **Run the set of 8 queries at the end of Homework4.pdf**
  - save the top ten results for each of the eight queries
9. **compare the two sets of results and describe any overlap**

# Step 1: Ubuntu with VirtualBox

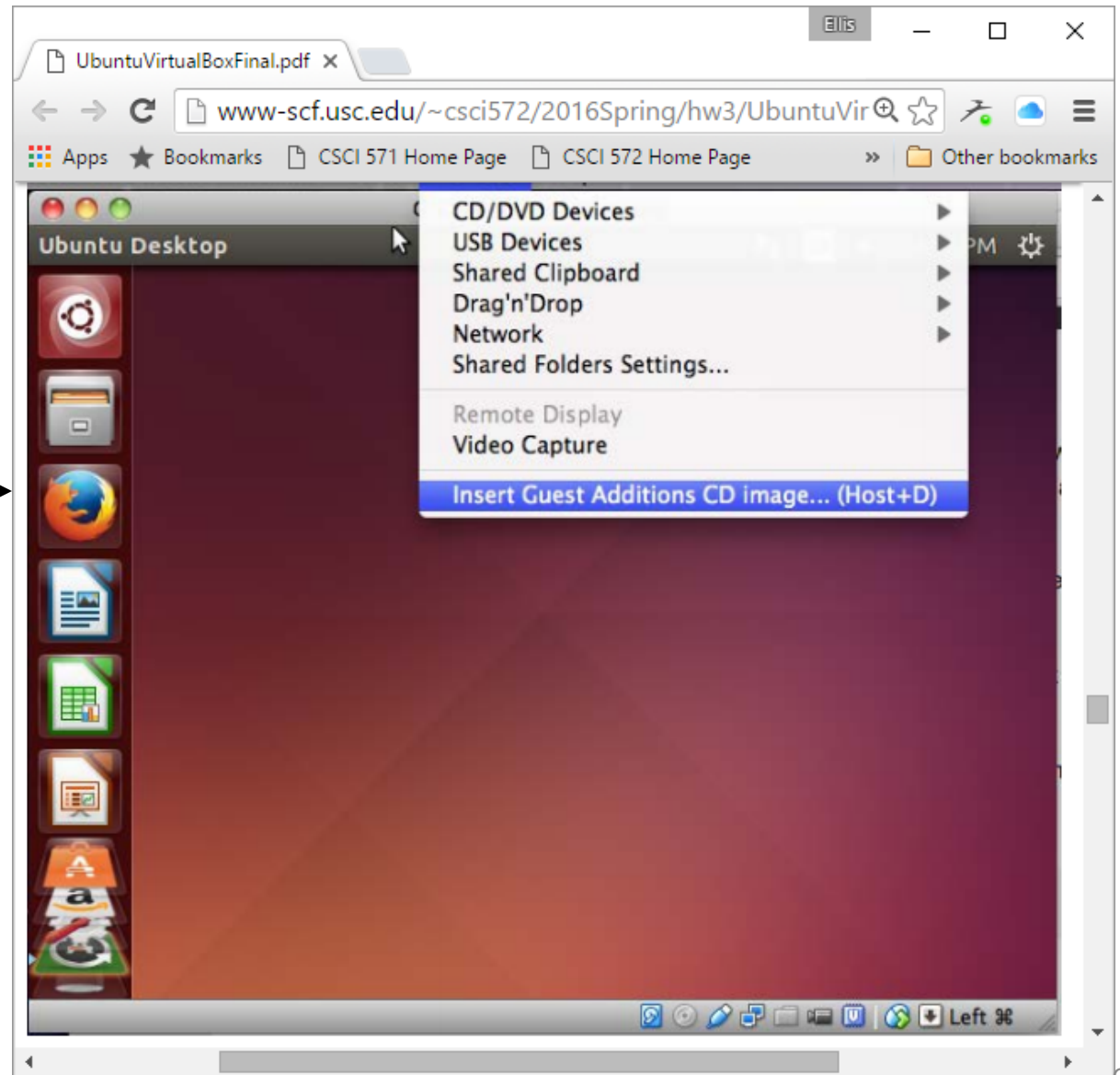
- **VirtualBox** is an open source, freely available Windows application (it also runs on other platforms) that lets you run multiple operating systems on your single machine
  - E.g. run Windows on a Mac, run Linux on Windows
  - Major supported operating systems include: Windows NT 4.0, Windows 2000, Windows 8, Windows 10, DOS Windows 3.x, Linux, Solaris, FreeBSD, OpenBSD
- **Ubuntu** is a Linux-based operating system distributed on personal computers, smartphones and network servers. It uses **Unity** as its default desktop environment
- **Solr requires a Unix environment** to run, so step 1 is required if you plan to use your Windows laptop

# Step 1: Setting Up Ubuntu with VirtualBox

1. Download the free version of VirtualBox for Windows machines
  - Instructions can be found here  
<http://www-scf.usc.edu/~csci572/2017Spring/hw4/UbuntuVirtualBoxFinal.pdf>
2. Download the Ubuntu 64-bit version
3. Run VirtualBox and select your Ubuntu version as the New Application
4. Set various parameters
5. Install Ubuntu and you should be ready to run

# Your Ubuntu/Unity Desktop

Built-in applications  
including  
Firefox browser



## Step 2: Installing Solr

- Solr is an open source enterprise search server based on the Lucene Java search library
- Instructions for downloading and installing Solr can be found here
  - <http://lucene.apache.org/solr/quickstart.html>
- Fast, high performance, scalable search/information retrieval library
- Initially developed by Doug Cutting (Also author of Hadoop)
- it provides for Indexing and Searching of documents
- produces an Inverted Index of documents
- Provides advanced Search options like **synonyms**, **stopwords**, based on **similarity**, **proximity**.
- **<http://lucene.apache.org/> is the main page for both Lucene and Solr**

Apache Lucene - Welcome x

lucene.apache.org

Apps ★ Bookmarks CSCI 571 Home Page CSCI 572 Home Page USC Computer Science D... USC ITS - Software >> Other bookmarks

Search with Apache Solr @ select provider

Lucene™

CORE (JAVA) SOLR PYLUCENE

Ultra-fast Search Library and Server

Lucene Solr

Apache Lucene and Solr set the standard for search and indexing performance

# Welcome to Apache Lucene

The Apache Lucene™ project develops open-source search software, including:

- **Lucene Core**, our flagship sub-project, provides Java-based indexing and search technology, as well as spellchecking, hit highlighting and advanced analysis/tokenization capabilities.
- **Solr™** is a high performance search server built using Lucene Core, with XML/HTTP and

lucene.apache.org/#

**DOWNLOAD** Apache Lucene 5.5.0

**DOWNLOAD** Apache Solr 5.5.0

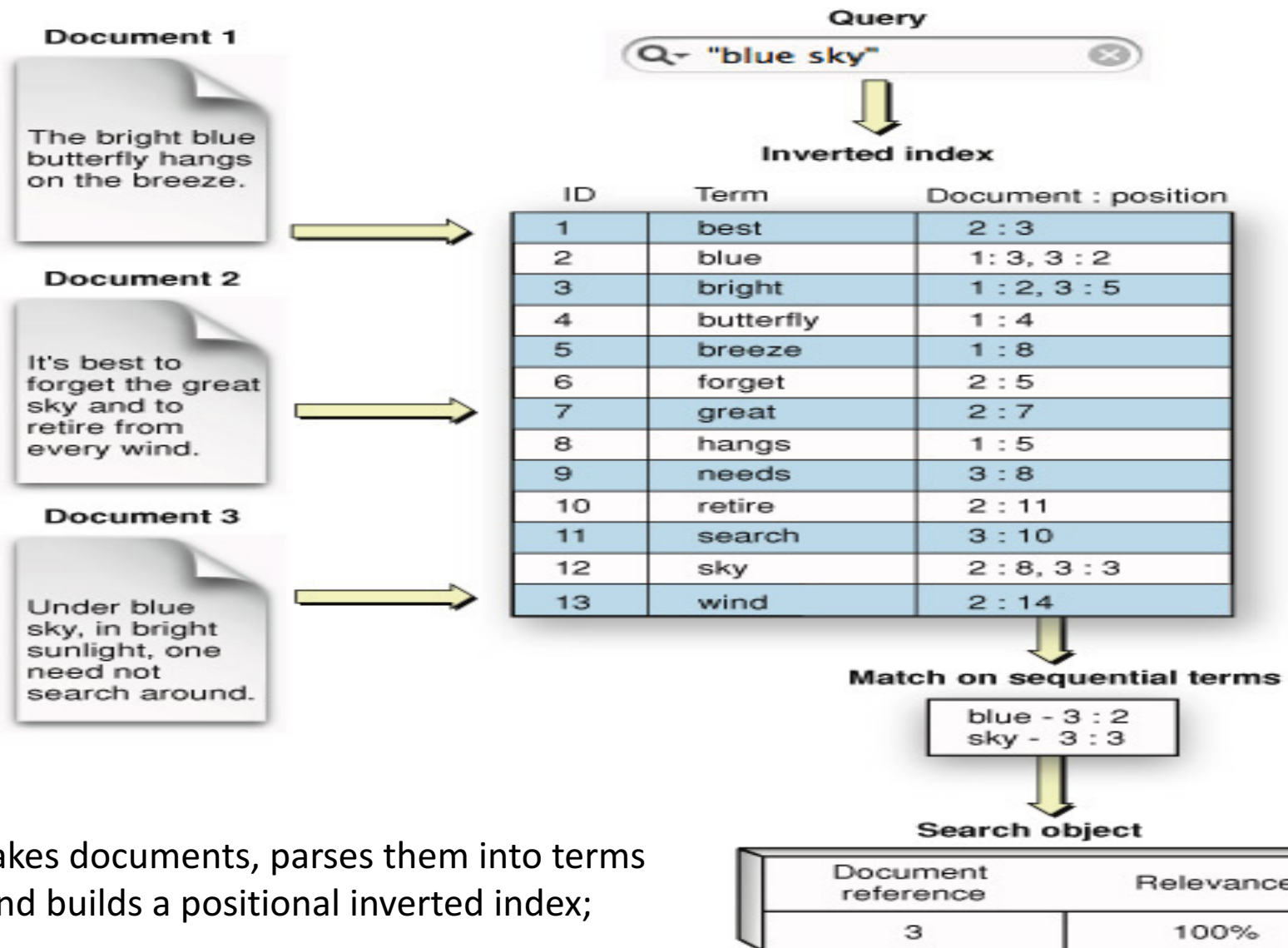
Download Lucene

Download Solr

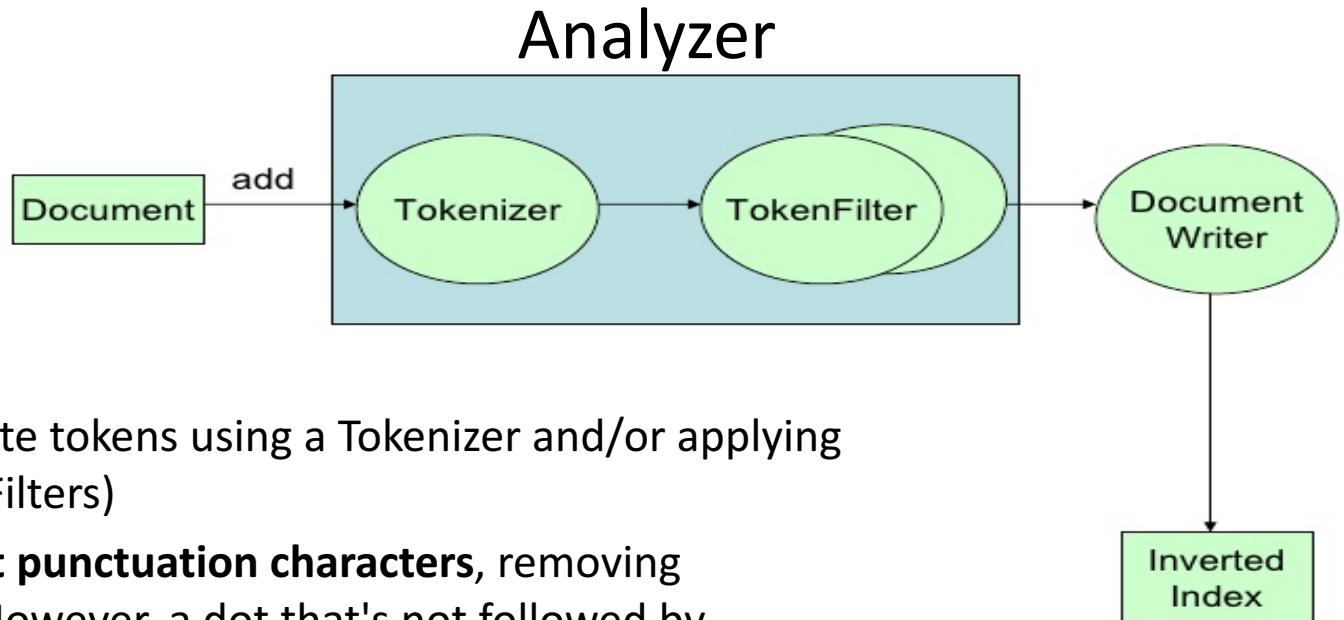
You only need to download Solr



# Lucene Internals - Positional Inverted Index



# Lucene Indexing Pipeline



- Analyzer : create tokens using a Tokenizer and/or applying Filters (Token Filters)
- **Splits words at punctuation characters**, removing punctuation. However, a dot that's not followed by whitespace is considered part of a token.
- **Splits words at hyphens**, unless there's a number in the token, in which case the whole token is interpreted as a product number and is not split.
- **Recognizes** email addresses and internet hostnames as one token.

# Lucene Scoring Concepts

## TF - IDF

Lucene scores using a combination of TF-IDF and vector closeness

$$w_{x,y} = tf_{x,y} \times \log \left( \frac{N}{df_x} \right)$$

**TF-IDF**

Term  $x$  within document  $y$

$tf_{x,y}$  = frequency of  $x$  in  $y$

$df_x$  = number of documents containing  $x$

$N$  = total number of documents

**TF - IDF** = Term Frequency **X** Inverse Document Frequency

cosine-similarity(query\_vector, document\_vector) =  $V(q) * V(d) / |V(q)| * |V(d)|$   
where  $V(q) * V(d)$  is the dot product of the weighted vectors and  $|V(q)|$ ,  $|V(d)|$  are the Euclidean norms of the vectors (square root of the sum of squares)

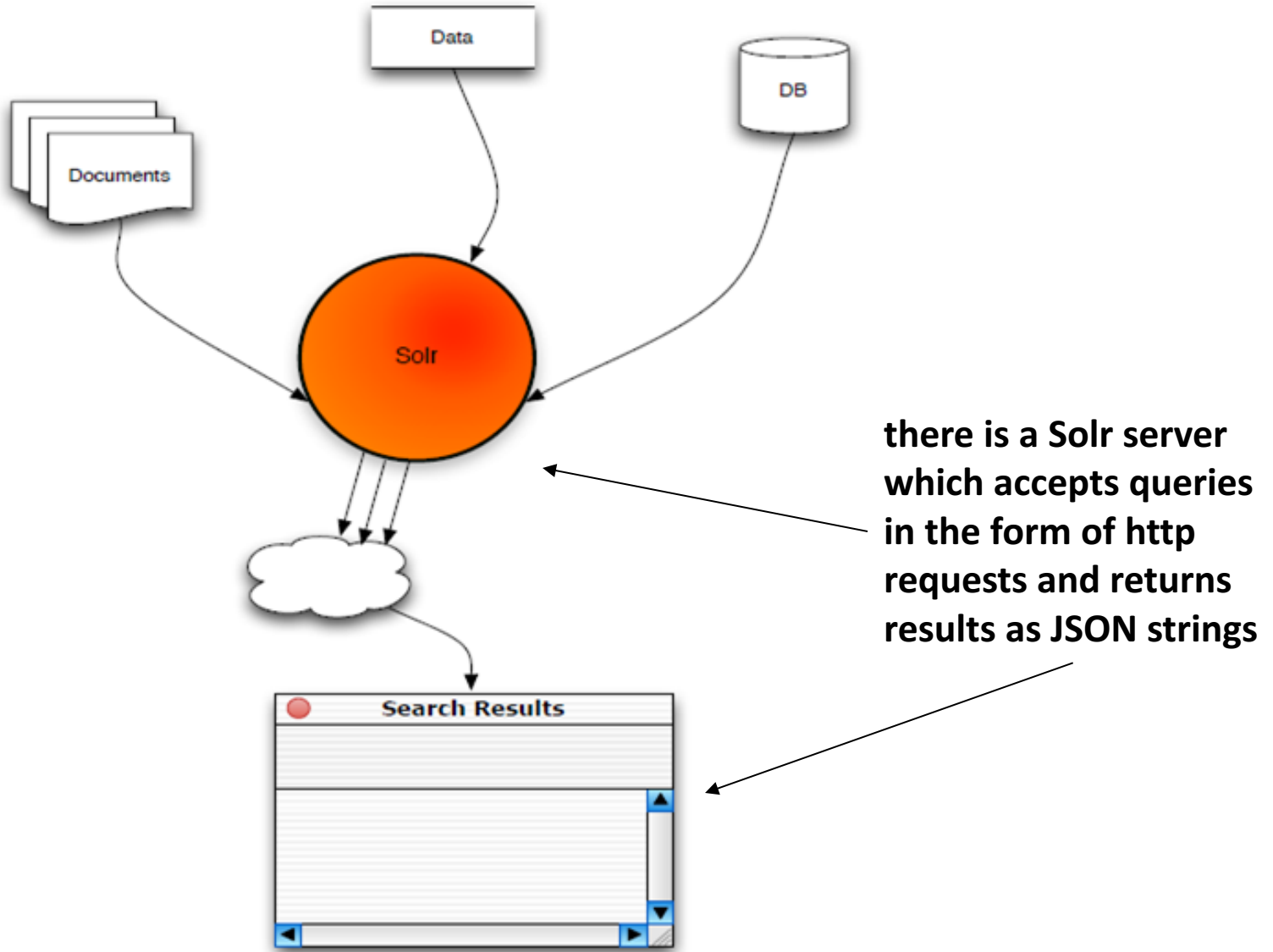
for details see

[https://lucene.apache.org/core/4\\_0\\_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html](https://lucene.apache.org/core/4_0_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html)

# Apache Solr

- Created by Yonik Seeley for CNET
- Enterprise Search platform for Apache Lucene
- Open source
- Highly reliable, scalable, fault tolerant
- Support distributed Indexing (SolrCloud), Replication, and load balanced querying
- <http://lucene.apache.org/solr>

# High level overview



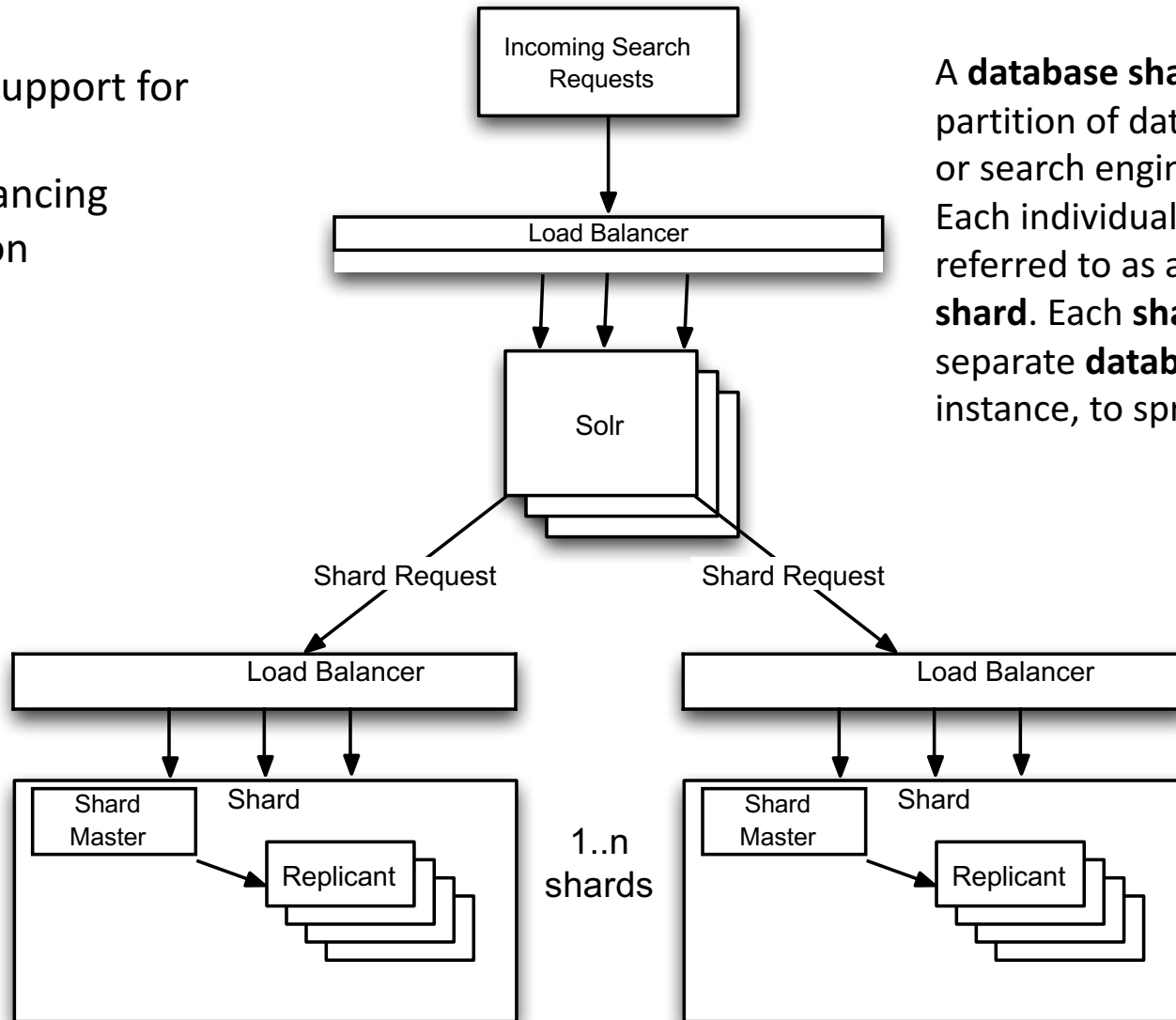
# Solr in Production

Includes support for

- Sharding
- Load balancing
- replication

A **database shard** is a horizontal partition of data in a **database** or search engine.

Each individual partition is referred to as a **shard**. Each **shard** is held on a separate **database** server instance, to spread the load.



# How to Start Solr

**Complete installation instructions can be found at**

<http://lucene.apache.org/solr/quickstart.html>

**Once it is installed:**

## **1. Start Solr**

```
java -jar start.jar
```

## **2. Index your data**

```
java -jar post.jar *.xml
```

## **3. Search <http://localhost:8983/solr>**

localhost indicates the Solr server is running locally on port 8983

# Querying Data

- HTTP GET or POST with parameters are used to specify queries
- E.g. here are 4 sample queries, some with various parameters

`http://solr/select?q=electronics`

`http://solr/select?q=electronics&sort=price+desc`

price and desc are fields

`http://solr/select?q=electronics&rows=50&start=50`

`http://solr/select?q=electronics&fl=name+price`

limit results to fields:  
name and price



# Querying Data: Results

- The standard response format is XML
- JSON is often used as well

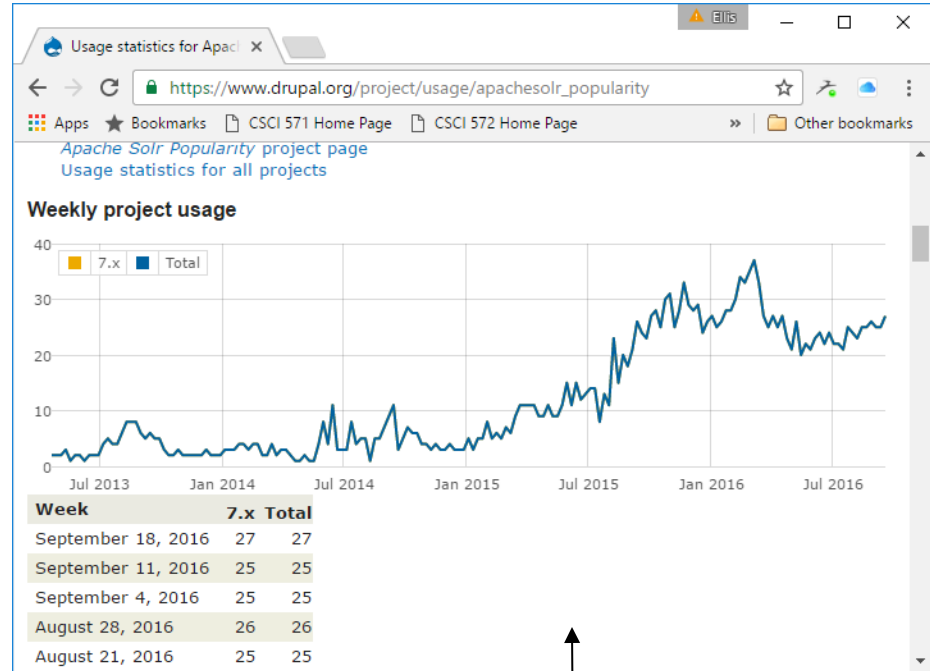
```
<response>
  <lst name="responseHeader">
    <int name="status">0</int>
    <int name="QTime">1</int>
  </lst>
  <result name="response" numFound="14" start="0">
    <doc>
      <arr name="cat">
        <str>electronics</str>
        <str>connector</str>
      </arr>
      <arr name="features">
        <str>car power adapter, white</str>
      </arr>
      <str name="id">F8V7067-APL-KIT</str>
      ...
    </doc>
  </result>
</response>
```

# Query Types

- Single and multi-term queries
  - ex fieldname:value or title: software engineer
- +, -, AND, OR NOT operators are supported
  - ex. title: (software AND engineer)
- Range queries on date or numeric fields,
  - ex: timestamp: [ \* TO NOW ] or price: [ 1 TO 100 ]
- Boost queries:
  - e.g. title:Engineer ^1.5 OR text:Engineer
- Fuzzy search : is a search for words that are similar in spelling
  - e.g. roam~0.8 => noam
- Proximity Search : with a sloppy phrase query. The close together the two terms appear, higher the score.
  - ex “apache lucene”~20 : will look for all documents where “apache” word occurs within 20 words of “lucene”

# Solr is Used by Many Projects

- **Search Engine**
  - Yandex.ru, DuckDuckGo.com
- **News Paper**
  - Guardian.co.uk
- **Music/Movies**
  - Apple.com, Netflix.com
- **Events**
  - Stubhub.com, Eventbrite.com
- **Cloud Log Management**
  - Loggly.com
- **Others**
  - Whitehouse.gov
- **Jobs**
  - Indeed.com, Simplyhired.com, Naukri.com
- **Auto**
  - AOL.com
- **Travel**
  - Cleartrip.com
- **Social Network**
  - Twitter.com, LinkedIn.com, mylife.com



Tracking the popularity of Solr

# Solr Includes Spell Checking

(you will need this for homework #4)

- Not enabled by default, see example config to wire it in
- <https://cwiki.apache.org/confluence/display/solr/Spell+Checking>
- File or index-based dictionaries for spell correction
- Supports pluggable distance algorithms:
  - Levenstein alg: [https://en.wikipedia.org/wiki/Levenshtein\\_distance](https://en.wikipedia.org/wiki/Levenshtein_distance)
  - JaroWinkler alg: ,  
[https://en.wikipedia.org/wiki/Jaro%E2%80%93Winkler\\_distance](https://en.wikipedia.org/wiki/Jaro%E2%80%93Winkler_distance)
- <http://wiki.apache.org/solr/SpellCheckComponent> is a full discussion of the spell checking abilities of Solr

# Solr Includes Autosuggestion

(you will need this for homework #5)

Enter your keywords:

teach

Did you mean: **teaching**

|                |    |
|----------------|----|
| <b>teach</b>   | 17 |
| teachers       | 2  |
| teacher        | 1  |
| teach book     | 15 |
| teach world    | 11 |
| teach wide     | 11 |
| teach teaching | 9  |
| teach computer | 9  |

Find **dinn**

|                             |
|-----------------------------|
| <b>dinner</b>               |
| <b>dinner</b> restaurant    |
| <b>dinner</b> and drinks    |
| <b>dinner</b> cruise        |
| <b>dinner</b> and dancing   |
| <b>dinner</b> date          |
| <b>dinner</b> theater       |
| <b>dinner</b> show          |
| <b>dinner</b> buffet        |
| <b>dinner</b> and live jazz |

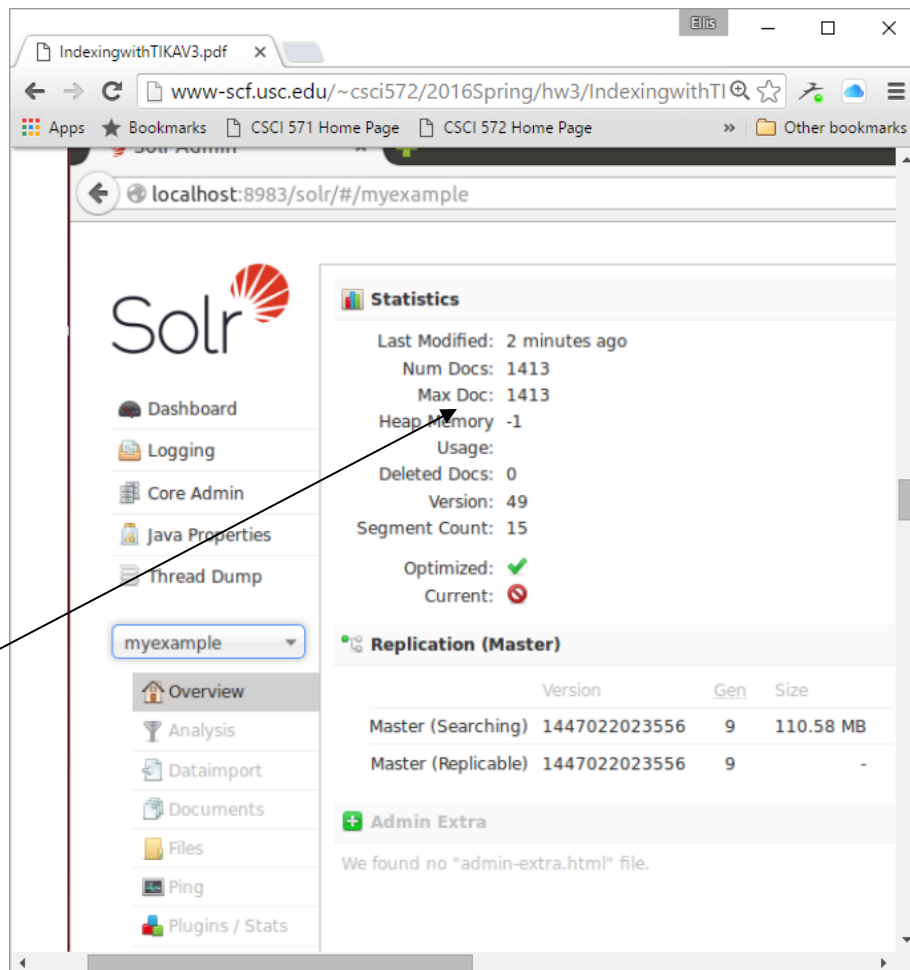
<https://wiki.apache.org/solr/Suggester>

# Downloading Your Data Set of Web Pages

- We have created a reference set of web pages for all news websites and placed them on Google drive
- Below is a URL that will give you access to these data sets
- The files are:
  - NYTimesData.zip
  - NBCNewsData.zip
  - LATimesData.zip
  - CNNDData.zip
  - ABCNewsData.zip
- Download only the zip file you are responsible for
- here is the URL
  - <https://drive.google.com/drive/u/1/folders/0B1xPYK9J4gtKSTg4b0d2Qm1DLWM>

# Step 3: Use Solr to Index a Web Site

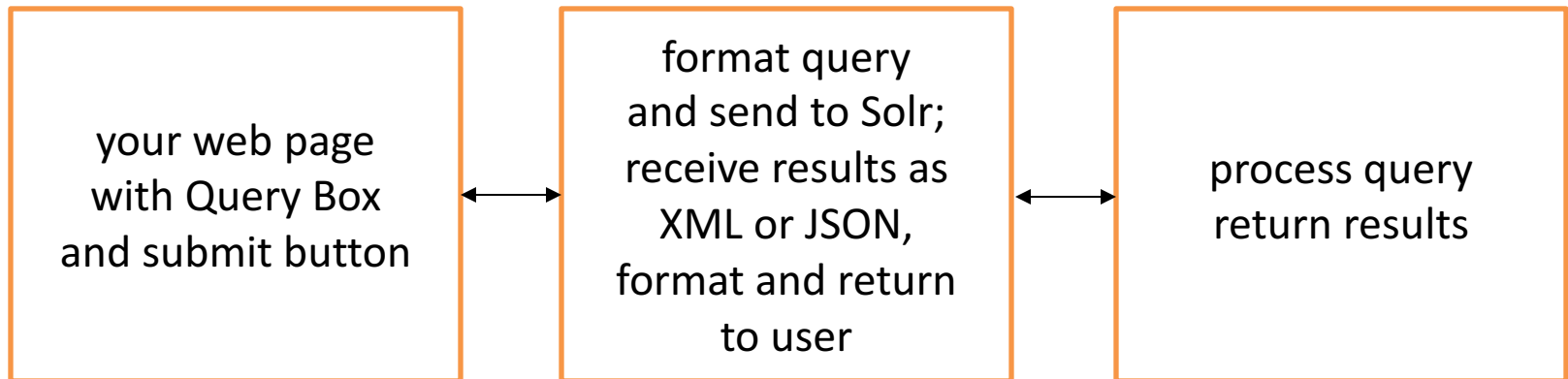
- start the Solr server
- start a new Solr core
- Use Tika to import your saved files
- Use the Solr interface to check that the files have been properly indexed
- Note the URL:  
localhost:8983/solr/#/myexample
- 1413 docs successfully indexed



# Step 4: The Actual Exercise

## Comparing Search Engine Ranking Algorithms

1. You should download the reference files for the news website you are responsible for;
2. You should install Solr as described previously;
3. Take the web pages from the reference files and index them in Solr, as described in earlier slides
4. Build a front end to Solr that permits a visitor to enter a query and get matching results
5. Solr will return the results in JSON format; your server needs to take the results and format them for the user



Web Browser

Your Web Server

Your version of Solr



# Apache Solr Client APIs

- http is the protocol to be used between client applications and Solr
- Clients use Solr's five basic operations: query, index, delete, commit, and optimize
- JavaScript is the standard client API and there is no package to be installed
  - http requests can be sent using XMLHttpRequest
  - Solr will respond with JSON output
- there is an output format specifically for Python
  - See <https://cwiki.apache.org/confluence/display/solr/Using+Python>
- Full list of Solr API clients can be found here
  - <https://cwiki.apache.org/confluence/display/solr/Client+API+Lineup>

## Step 4: The Actual Exercise

Here is a PHP client that accepts input from the user in a HTML form, and sends the request to the Solr server. After the Solr server processes the query, it returns the results which is parsed by the PHP program and formatted for display

```
<?php
```

```
// make sure browsers see this page as utf-8 encoded HTML
```

**returning a web page**

```
header('Content-Type: text/html; charset=utf-8');
```

```
$limit = 10;
```

**test for a query**

```
$query = isset($_REQUEST['q']) ? $_REQUEST['q'] : false;
```

```
$results = false;
```

```
if ($query)
```

**this is the Solr client library**

```
{ require_once('Apache/Solr/Service.php');
```

```
// create a new solr service instance - host, port, and corename
```

```
// path (all defaults in this example)
```

**Solr runs on port 8983**

```
$solr = new Apache_Solr_Service('localhost', 8983, '/solr/core_name/');
```

```
// if magic quotes is enabled then stripslashes will be needed
```

```
if (get_magic_quotes_gpc() == 1)
```

**handles quoting of special characters in query**

```
{ $query = stripslashes($query); }
```

## PHP Program (2 of 3)

```
try
{ $results = $solr->search($query, 0, $limit); }
catch (Exception $e)
{ die("<html><head><title>SEARCH EXCEPTION</title><body><pre>{$e-
>__toString()}</pre></body></html>"); } }
?>
<html> <head> <title>PHP Solr Client Example</title> </head> <body>
<form accept-charset="utf-8" method="get">
<label for="q">Search:</label>
<input id="q" name="q" type="text" value="<?php echo htmlspecialchars($query, ENT_QUOTES,
'utf-8'); ?>"/>
<input type="submit"/>
</form>
<?php
// display results
if ($results)
{ $total = (int) $results->response->numFound; $start = min(1, $total); $end = min($limit, $total);
?>
<div>Results <?php echo $start; ?> - <?php echo $end;?> of <?php echo $total; ?>:</div>
<ol>
```

← **send query to Solr**  
← **catch any exception**

← **create web page output**  
**create input text box**

← **create submit button**

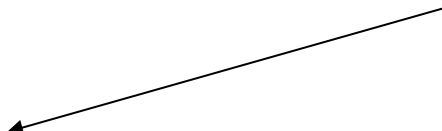
**end form**

← **JSON result string**

## PHP Program (3 of 3)

```
<?php
// iterate result documents
foreach ($results->response->docs as $doc)
{ ?> <li>
  <table style="border: 1px solid black; text-align: left">
<?php
// iterate document fields / values
foreach ($doc as $field => $value)
{ ?>
  <tr>
    <th><?php echo htmlspecialchars($field, ENT_NOQUOTES, 'utf-8'); ?></th>
    <td><?php echo htmlspecialchars($value, ENT_NOQUOTES, 'utf-8'); ?></td>
  </tr>
<?php }
?> </table> </li>
<?php } ?>
</ol>
<?php
} ?> </body> </html>
```

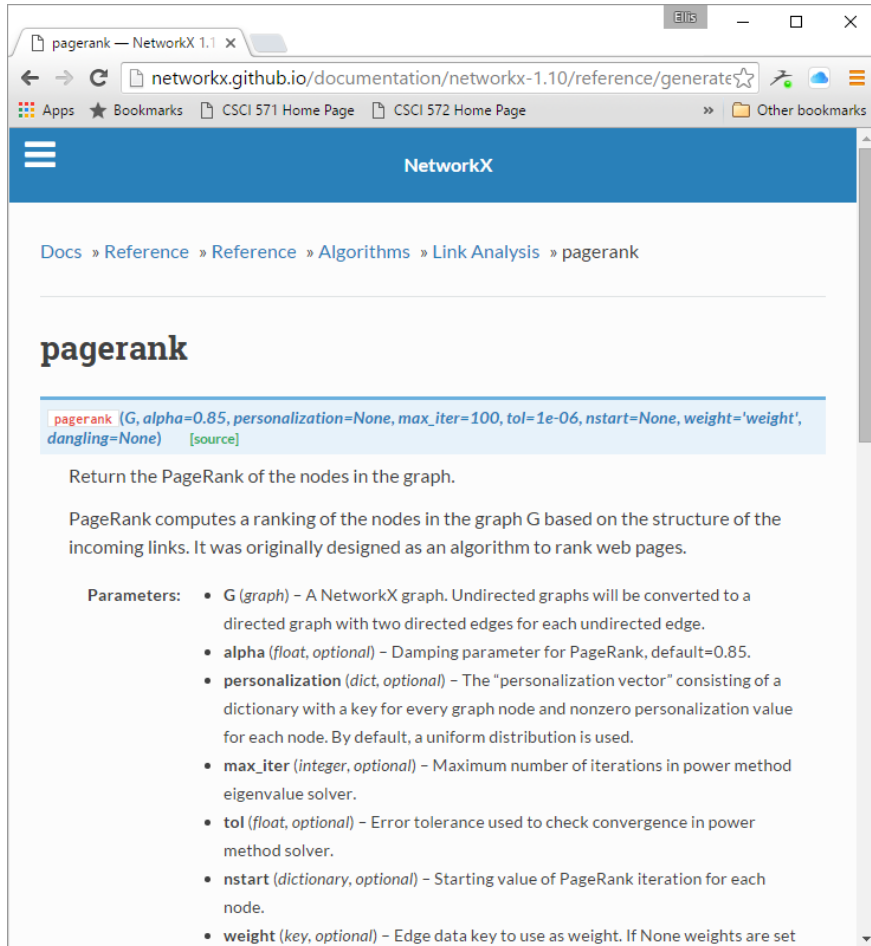
output consists of a  
set of field, value  
pairs



# Comparing Ranking Algorithms

- we have already seen the built-in Solr ranking method
  - see slide 11
- Solr permits alternative ranking algorithms
  - to use PageRank we must create pageranks for all web pages
  - to do this we use an open source program called NetworkX
  - we need to create the web graph (pages and links) so the PageRank can be determined
  - See the following document for a method on extracting links and producing the NetworkX file:
  - <http://www-scf.usc.edu/~csci572/2017Spring/hw4/ExtractingLinks.pdf>

[http://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.link\\_analysis.pagerank\\_algorithm.html](http://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.link_analysis.pagerank_algorithm.html)



- You are going to use an open source PageRank algorithm, networkx, located at URL above;
- networkx is a Python program
- You need to create a graph that the PageRank algorithm can work on

Important Parameters:

- a NetworkX graph
- a damping parameter (e.g. 0.85)
- maximum number of iterations
- error tolerance
- starting Page Rank value of nodes

[http://networkx.github.io/documentation/networkx-1.10/\\_modules/networkx/algorithms/link\\_analysis/pagerank\\_alg.html#pagerank](http://networkx.github.io/documentation/networkx-1.10/_modules/networkx/algorithms/link_analysis/pagerank_alg.html#pagerank)

### **Source code for networkx.algorithms.link\_analysis.pagerank\_alg**

```
"""PageRank analysis of graph structure. """
```

```
def pagerank(G, alpha=0.85, personalization=None,
```

```
            max_iter=100, tol=1.0e-6, nstart=None, weight='weight', dangling=None):
```

```
    Return the PageRank of the nodes in the graph.
```

#### **Parameters**

*G* : *graph*: A NetworkX graph.

*alpha* : float, optional Damping parameter for PageRank, *default*=0.85.

*personalization*: dict, optional; By *default*, a uniform distribution is used.

*max\_iter* : integer, optional Maximum number of iterations in power method eigenvalue solver.

*tol* : float, optional; Error tolerance used to check convergence in power method solver.

*nstart* : dictionary, optional Starting value of PageRank iteration for each node.

*weight* : key, optional; If None weights are set to 1.

#### **Returns**

*pagerank* : dictionary; Dictionary of nodes with PageRank as value

#### **Examples**

```
>>> G = nx.DiGraph(nx.path_graph(4))
```

```
>>> pr = nx.pagerank(G, alpha=0.9)
```

# Some NetworkX Operations

- Create an empty graph with no nodes and no edges.

```
>>> import networkx as nx
```

```
>>> G=nx.Graph()
```

- add one node at a time

```
>>>G.add_node(1)
```

- add a list of nodes

```
>>>G.add_nodes_from([2,3])
```

- add one edge at a time

```
>>>G.add_edge(1,2)
```

```
>>>e=(2,3)
```

```
>>>G.add_edge(*e) #unpack edge tuple*
```

- add a list of edges

```
>>>G.add_edges_from([1,2),(1,3)])
```



# Final Steps

- Input to the PageRank algorithm a file containing every document ID and associated with each ID, the IDs that are pointed to by links within the document ID
- Output from the PageRank algorithm is a file containing every document ID and its associated PageRank
- place this file in solr-5.3.1/server/solr/core\_name, call the file external\_pageRankFile.txt
- add the PageRank field to the schema.xml file

```
<fieldType name="external" keyField="id" defVal="0" class="solr.ExternalFileField" valtype="pfloat" />  
<field name="pageRankFile" type="external" stored="false" indexed="false" />
```

# The Queries

- Once both ranking algorithms are working you should input the queries below and compare the results

| Query               |
|---------------------|
| Brexit              |
| NASDAQ              |
| NBA                 |
| Snapchat            |
| Illegal Immigration |
| Donald Trump        |
| Russia              |
| NASA                |