Brownie Waffles:
Programmatically Accelerating Human Creativity with Cross-Discipline Connections

Cameron Bell

Advisor: Dr. Dan Ventura, Computer Science

April 23, 2016

Physics 492R Capstone Project Report

Table of Contents

Abstract:

Creativity is a process of connection, not of synthesis. Great ideas are characterized by the combination of known concepts - for example, brownie waffles. The genius is neither in the brownies nor the waffles, but in the combination of the two. Theoretically, a computer could make the connections for us, researching and suggesting other products, industries, and fields whose methods could be of use in our processes of development. Using latent semantic indexing, we have mathematically mapped semantic relationships among the documents on Wikipedia and developed a tool that compares documents, allowing us to work on finding these valuable creative connections, or *intuitive leaps*. Though this could be done much faster with existing technologies like Google, our model needed to be fully customized for specific applications.

Introduction:

Nearly half of all academic research papers are never cited again, even by their authors in later studies (Redner). This doesn't mean, however, that half of academic research is worthless. It only shows the limitations of the human mind to access and read every bit of information available. But what if, with the aid of a computer, someone could browse all the available literature on a topic in an instant? Surely he or she would make brilliant connections, generating ideas that could change the world.

At its heart, creativity is a process of connection (Simonton). Artists don't have to discover paint in order to create a masterpiece; they merely combine existing colors and mediums in a unique way. Great works of literature are written with existing words -- the combination of them is what makes the work great. In the Information Age, facts and ideas are at our fingertips, yet we fail to fully utilize them. With the aid of a computer, however, the collective knowledge and intelligence of humankind could be harnessed, catapulting creativity further than ever before.

It may seem far-fetched to think that a computer could possibly fake the incredible creative capacities of the human mind, but it has already proven to enhance the brain in many arenas, including chess. The most skilled chess player is neither a human nor a computer, but a team of human and computer together (Kelly). The discretion and decision-making skills of a human are especially powerful when paired with the rapid learning, researching, and analytic capabilities of a computer.

But even the computer alone can be impressively creative. A team of computer scientists at Brigham Young University developed a computer program called PIERRE (Pseudo-Intelligent Evolutionary Real-time Recipe Engine) that researches databases of recipes and generates

new ones on its own. It has successfully produced tasty and original recipes by itself (Morris, et. al).

We hope to aid human users in the process of making *intuitive leaps*. This means connecting the dots by finding existing concepts and products that can help in the development of another. The computer does this by researching and performing analysis on Wikipedia articles, which provide a consistently formatted encyclopedia of topics.

Methods:

Obtaining and Organizing the Data

Because Wikipedia is publicly available on the internet, we can access the content of the articles to perform our analysis. To aid in this process, we used Wikipedia “dumps” from the Wikimedia Foundation. The one we used is a compressed file of about 12 MB that contains the text of all the English articles on Wikipedia. The Wikimedia foundation is a fantastic source, with a mission “to create educational content that is freely available to all people.” Because the content can be shared freely, we didn’t have any copyright concerns. Here is the link to the dump we used: https://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2

From that wikipedia dump, we used an open-source python program called “wikiextractor” (Attardi) to convert from the .xml.bz2 format to a collection of text files, each one housing the text of around 30-40 articles. Within the text file, these articles were organized with HTML tags, making it easy to access the title of each article and its text fairly easily. Because our original approach involved web scraping from Google searches, I was familiar with an HTML parsing library called *lxml* in Python, offering easy access to the data. Below we will discuss why this may be what is causing the analysis to take so long.

Mapping the data mathematically

Now that the data was arranged in a useful format (which I thought was easily accessible), the next step was the actual analysis. This is where my understanding of physics principles helped the most in developing a useful model. Most of our analysis was done using a Python package called *gensim* (Rehurek), which was built for natural language processing. The remainder of this section will describe the steps taken to convert each article into a vector in LSI space. Many of these steps were done with gensim. We began by cleaning the text of each article to make it suitable for analysis, then formed a comprehensive dictionary, vectorized each article, applied some transformations to the resulting vectors, and finally performed similarity queries on our user-input ideas.

With the text in HTML format, it was ready to be cleaned in a way that would allow it to be analyzed mathematically. First, each text file was read into python and separated into individual articles using the *lxml* library. Each article was then *tokenized*, or broken up into a list of words. From this list of tokens we removed a set of words called “stopwords,” or words that are so common that they don’t indicate the topic of the document. These are words such as a, an, the, and, of, etc. This list was obtained from the natural language toolkit in Python, through the *nltk* library. Each of the words was then trimmed to a standard base root - this process is called “stemming” (also from *nltk*). For example, the words *running, ran, runs,* and *runner* would all become *run.* This ensures that the analysis looks at prevalent themes instead of focusing on the different individual word forms in an article.

After the text was cleaned, we used gensim to iterate over all of the documents to make a dictionary - this took 4 days, 18 hrs, and 15 min on the BYU supercomputer. A gensim dictionary is a python object that assigns a number to the 100,000 most common word stems. This makes it easier to perform analyses by ordering all document statistics into a matrix.

Because all of the word stems were assigned a numerical value, we could easily create a matrix of statistics for each document.

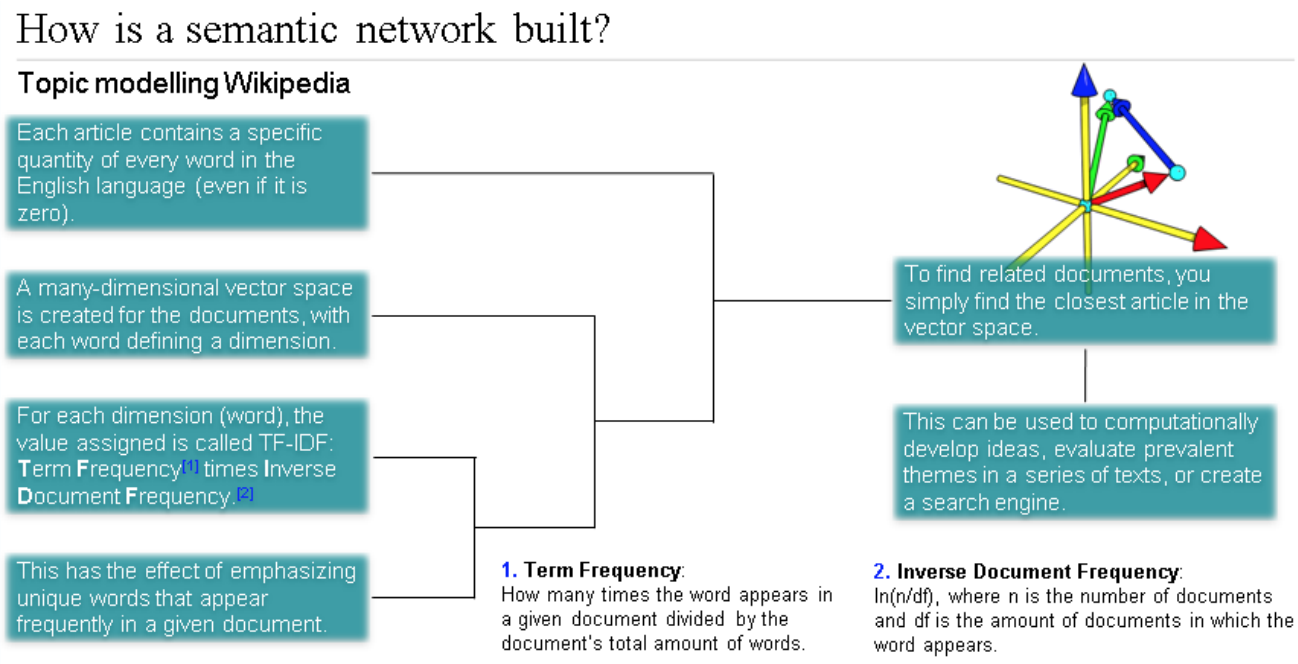
This matrix is called a “bag-of-words.” The reason is that the order of the words is no longer recorded after a document is converted to a matrix. It is represented as 100,000 values, each of which represents a specific word stem (the position in the matrix) and a number for how many times it appears in the document. For example, suppose the dictionary assigned the number 635 to the word *run*. Then for a given document that uses forms of *run* 27 times, the 635th position in its corresponding bag-of-words vector would hold the value 27. Of course, with 7,750,000 documents, it would take a lot of memory to store 100,000 values for each article. Because each article only uses a small subset of the 100,000 words, its information is stored as a *sparse matrix*, or one which only records the non-zero values. These sparse matrices can still be used to perform analyses on the documents. We saved the sparse vectors in *matrix market format,* which can be read about here (NIST). This collection of vectors is called a *corpus*.

Forming a bag-of-words matrix for each document took 3 days, 17 hrs, and 33 min, after which we were prepared to represent each document as a vector in a 100,000-dimensional vector space. Though the computer doesn’t have to visualize these matrices as vectors in order to perform useful analyses, it can be helpful to us to visualize vectors in a 3-dimensional space in order to understand the methods we used. The bag-of-words format is not the most useful format for comparing documents; the raw statistics alone do not allow us to reliably find documents that are related by similar topics. For example, a document about nuclear energy may be tied to one about toilet plungers just because both involve obtaining patents. Though the topics might both use terminology referring to the patent process, we are not necessarily interested in that. We are *more* interested in the actual technology. In order to emphasize the *unique* aspects of each document, we converted the bag-of-words vectors to *term frequency -*

inverse document frequency vectors (or tf-idf vectors). This means that each value is no longer simply the number of times a word appears, but a tf-idf value:

$$tf * idf = \frac{frequency_{token}}{doc\ length} * ln(\frac{n_{total}}{n_{containing}})$$

, where $frequency_{token}$ is the number of times a given token appears in the document, doc length is the total number of words in the document, n_{total} is the total number of documents in the corpus, and $n_{containing}$ is the number of documents that contain that token. It took only 2 hrs 38 min to convert the bag-of-words corpus to tf-idf space. Figure [1] explains how the process works.



Sources: http://en.wikipedia.org/wiki/Semantic_network,

http://recommender-systems.org/vector-space-model/,

http://www.cgplusplus.com/online-reference/maxscript-reference/source/how_do_i_make_a_vec

tor_from_a_vertex_position_.htm

For each document, the tf-idf vector representation emphasizes the unique aspects of the document's topic. Term Frequency increases the document's value for a particular term (and corresponding dimension) if it is used many times, and Inverse Document Frequency increases the value if the term is not used in many other documents. The vectors are then normalized to become unit vectors (their absolute value is 1). Two problems with this model are synonymy and polysemy. Some documents use different words that mean the same thing in order to discuss the same concept, which gets lost in the tf-idf vector space model (synonymy). Polysemy is the opposite problem: two documents may use the same word with different meanings to discuss different topics, but the documents are still linked. Nonetheless, representing the documents in a tf-idf vector space allows for an easy transition to a different model: *latent semantic indexing*.

Latent semantic indexing analyzes the *structure* of documents as opposed to simple term frequency values. All of the tf-idf vectors can be combined into one matrix, where each row represents a unique token and each column represents a document. This enormous *m x n* matrix is called a *term-document matrix*, which we will call *A*. LSI then performs a rank-reduced singular value decomposition on the matrix *A*, essentially analyzing the structure (instead of specific features) of the documents. It links terms to concepts instead of just to specific documents. Here's how it works mathematically:

$$A \approx TSD^T$$

$$T^T T = I_r \quad D^T D = I_r$$

$$S_{1,1} \geq S_{2,2} \geq \ldots \geq S_{r,r} > 0 \quad S_{i,j} = 0 \text{ where } i \neq j$$ (Wikipedia)

What this means is that *A* is our *m x n* term-document matrix, decomposed into *T*, *S*, and D^T. *T* is an *m x r* term-topic matrix (where r is the lesser of m and n), *S* is an *r x r* diagonal matrix composed of descending singular values, and D^T is an *n x r* topic-document matrix. This allows us to extract information about topics based on the structure of the vectors.

This decomposition is then truncated, which means that only the largest singular values in *S* (and the corresponding values in *T* and D^T) are kept and all other values are discarded. This leaves only *k* values, where *k* << *r.* Now *T* is an *m x k* matrix, *S* is a *k x k* matrix, and D^T is an *n x k* matrix. The original *m x n* matrix *A* can now be represented this way:

$A \approx A_k = T_k \, S_k \, D_k^T$ This truncation process eliminates noise and emphasizes the most important characteristics of the corpus.

k is now the number of dimensions in the new vector space model. An LSI vector space is recommended to have 200-500 dimensions **[CITATION]**, each of which represents a topic. We created an LSI model with 400 topics, then transformed the tf-idf corpus into the new LSI space. What this means is that each document has a value corresponding to each topic, representing the prevalence of the topic in the document. This is why similar and synonymous words can be linked across documents and small errors and aberrations are easily accommodated, which is especially important when processing short documents.

Distributed Computing

Imagine a pile of 1 million bricks by the JSB. If the university assigned one worker to move the pile to the ASB, it would take him 1 million trips. However, if a team of 1,000 workers tackled the project, it would take only 1,000 trips. This is like a computer; if it shares the load of computing a process, it can go much faster. However, to create an LSI space, the computers need to communicate with each other. This is called *parallel computing*. This would be like an assignment to build a sculpture with the 1 million bricks. Many hands make light work, but because they must all know the design of the sculpture and the current status, at a certain point it no longer helps to have extra workers. They need a leader - this is why one of the computers in parallel computing is referred to as a *dispatcher* and the others are referred to as *workers.*

We created the LSI model serially (with 1 worker) and on parallel machines (with 1 dispatcher and 10 workers). The process ran almost 5 times faster!

Here are the times:

Serially: 11:23:01 to 5:48:11 (**6:25:10 total**)

Distributed: 17:42:49 to 19:02:23 (**1:19:24 total**)

Figure 2 shows the program *top* displaying the activity of the 11 assigned processors.

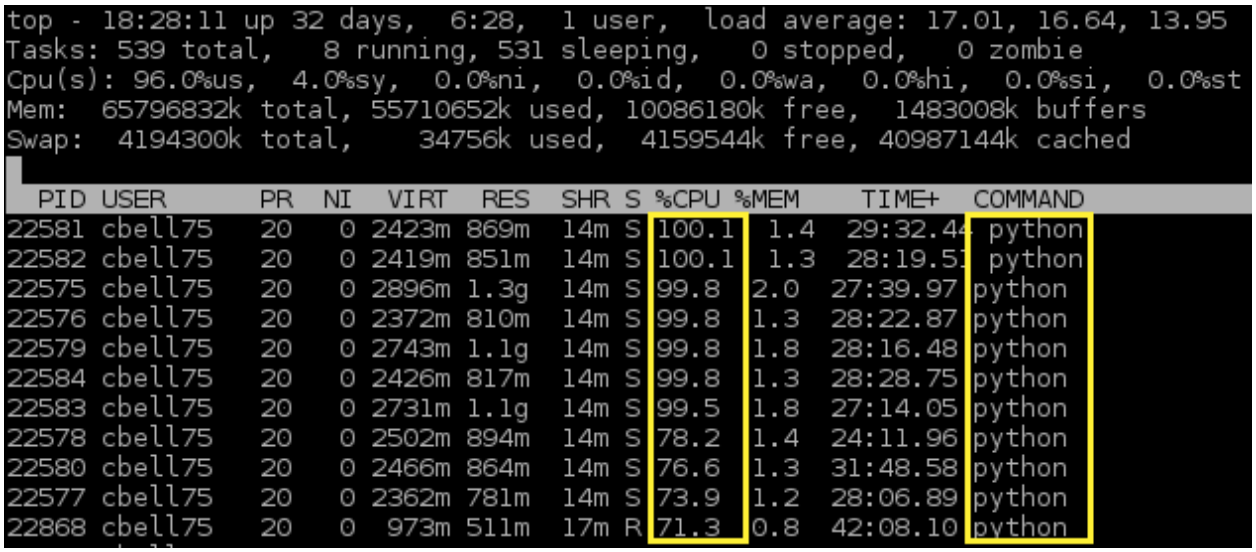


Figure 2 - Eleven processors running python

After the LSI space was defined, we converted our tf-idf corpus into LSI space so that we could perform similarity queries. This is done by loading a document into the previously defined LSI vector space (using a saved model and dictionary), then comparing with each of the documents in Wikipedia to find the closest ones. The distance is not simply a Euclidean distance (which uses the Pythagorean Theorem); instead, similarity is defined by angular distance.

Results

Because this was our first trial, we expected imperfect results. The program returns similar documents successfully (in about 30 seconds, which is practical for a web application) and seems to have more relevant results with longer documents. Interestingly, sometimes it returns documents with few shared words; instead, it finds documents similar in structure. Though this is the goal of latent semantic analysis, we did not expect it to work so well (this is not a desirable feature for our purposes). For example, when the following document was compared with the

Wikipedia corpus, the most similar document had no instances of UV, ultraviolet, IR, infrared, frequency, or wavelength.

Query document: “Sunblock typically refers to opaque sunscreen that is effective at blocking both UVA and UVB rays and uses a heavy carrier oil to resist being washed off. Titanium dioxide and zinc oxide are two of the important ingredients in sunblock.[48] Unlike the organic sun-blocking agents used in many sunscreens, these metal oxides do not degrade with exposure to sunlight. The use of the word "sunblock" in the marketing of sunscreens is controversial. Since 2013, the FDA has banned such use because it can lead consumers to overestimate the effectiveness of products so labeled.[41] Nonetheless, many consumers use the words sunblock and sunscreen synonymously. For total protection against damage from the sun, the skin needs to be protected from UVA, UVB and IRA (infrared light). Roughly 35% of solar energy is IRA.”

Most similar documents:
82.33 % similarity: Shielding gas
82.10 % similarity: Cryogenics
82.04 % similarity: Heptanal
81.91 % similarity: Graphite
81.88 % similarity: Silicone
81.59 % similarity: Silicon
81.46 % similarity: Fluorochemical industry
81.16 % similarity: 5083 aluminium alloy
80.86 % similarity: Monel
80.74 % similarity: Chlorotrifluoromethane
80.68 % similarity: Pyrotechnic initiator
80.56 % similarity: HPC catalysts
80.49 % similarity: Barium azide
80.35 % similarity: Magnetic refrigeration
80.34 % similarity: Octafluoropropane
80.17 % similarity: Cholesteryl chloride
80.16 % similarity: Gas mantle
80.06 % similarity: Chemical vapor deposition
80.02 % similarity: Dimethoxymethane
79.92 % similarity: Sterilization (microbiology)

These documents seem to be more similar to the scientific structure of the query document than they are in topic. Another example shows how this could be effective in brainstorming. Suppose a research and development team wants to make a rapid prototype, but they are not sure how to do it. After running this LSI similarity analysis on a technical description of rapid prototyping, this is the output:

Most similar documents:
88.89 % similarity: Computer simulation
85.99 % similarity: Computer-aided inspection
85.89 % similarity: Level of detail
85.51 % similarity: Surfel
85.18 % similarity: Solid modeling
84.72 % similarity: Wire-frame model
84.55 % similarity: Transistor model
83.59 % similarity: Phong shading
83.44 % similarity: Group method of data handling
83.44 % similarity: Parallel random-access machine
83.10 % similarity: Transdichotomous model
82.63 % similarity: Symbolic trajectory evaluation
82.46 % similarity: Leabra
82.41 % similarity: Uncertainty and errors in cfd simulation
82.19 % similarity: 3D modeling
82.18 % similarity: Polarizable continuum model
82.14 % similarity: Hierarchical database model
82.09 % similarity: Semiconductor process simulation
81.88 % similarity: X-machine
81.83 % similarity: Geometric design
81.60 % similarity: Turing machine equivalents
81.56 % similarity: Parasolid
81.48 % similarity: T-vertices
81.45 % similarity: Stanford dragon
81.40 % similarity: IBM Personal System/2
81.27 % similarity: Stanford bunny
81.17 % similarity: Baldwin–Lomax model
81.15 % similarity: Teletype Model 33
81.06 % similarity: Generalized linear mixed model
80.87 % similarity: Geometric modeling
80.83 % similarity: Fluid queue
80.67 % similarity: Dimensional modeling
80.65 % similarity: Computational neuroscience

Though this could be marginally useful, it is not the intended final product.

Applications

The previous example of querying a document about rapid prototyping could be carried one step further in order to be extremely useful. They could look into “computational neuroscience” (the last suggestion) and see how they model their problems. But suppose the team is designing a car. As it stands, they have a list of techniques that they could look to in

order to build their rapid prototype. The goal, however, is to find *other industries and fields* that are already using these technologies - we call this an *intuitive leap.* If the team were to know of specific products which used the same techniques and technologies that they plan on using, they could borrow from their successes. For example, sunblock aims to block specific wavelengths of light, and noise-cancelling headphones aim to block specific frequencies (wavelengths) of sound. This connection, a process of creativity, could lead to UV-cancelling (via destructive interference) devices. And, though this idea is impractical, it shows the potential power of such connections, such **intuitive leaps**. This will take some experimentation to figure out how to get these, but now we have the infrastructure implemented.

Some possible approaches:

- Use the tf-idf corpus to perform analyses or even directly on the bag-of-words corpus. This would allow us to look more at specific words (such as ultraviolet) as opposed to overall themes (such as sunburn).
- Ignore the closest matches, instead looking at some further down the list. Though we have not yet seen success with this, it could work if we tailor our search queries more carefully
- Try using LDA, another analysis model like LSI

When we figure out how to get intuitive leaps, we hope to tailor our search queries programmatically, so as to make the program more user-friendly. We also plan to evaluate the connections programmatically using the following methods:

- We replicated Columbia University's research on programmatic idea evaluation (Toubia, et. al). It analyzes the structure of human-produced ideas (by scraping Google searches) and compares it to computer-generated ideas. It is based on

the assumption that anything on the web was produced by a human, and, therefore, qualifies as a good comparison metric to machine-produced ideas.

- We are taking this a step further by also comparing the connections to ideas on kickstarter.com. The more funding, the better the idea. This would work well for only a specific type of idea (small, single-consumer products), but could be extremely useful when combined with rapid prototyping.

Another application of this latent semantic comparison analysis would be to upload a resume or LinkedIn profile. The program would return fields that align with a person's skills and interests. These results could then be plugged into Google and Google Trends. A high amount of results on Google would correlate with high supply, whereas a high amount of search queries on Google Trends would indicate high demand. The results with low supply and high demand would be returned, offering a possible entrepreneurial path.

Though the analysis itself may not be particularly ground-breaking, further work and research will hopefully accelerate the field of computer-aided creativity.

Works Cited

https://dumps.wikimedia.org/legal.html

Attardi, Giuseppe. https://github.com/attardi/wikiextractor

Kelly, Kevin. "The Future of AI? Helping Human Beings Think Smarter (Wired UK)." Wired UK. N.p., 03 Dec. 2014. Web. 08 Jan. 2016.

Morris, Richard G., Scott H. Burton, Paul M. Bodily, and Dan Ventura. "Soup Over Bean of Pure Joy: Culinary Ruminations of an Artificial Chef." International Conference on Computational Creativity (2012)

Redner, S. "How Popular Is Your Paper? An Empirical Study of the Citation Distribution." The European Physical Journal B Eur. Phys. J. B 4.2 (1998): 131-34. Web. 8 Jan. 2016.

Simonton, Dean Keith. "Creative Thought as Blind-variation and Selective-retention: Combinatorial Models of Exceptional Creativity." Physics of Life Reviews 7.2 (2010): 156-79. Web.

Rehurek, Radim. https://radimrehurek.com/gensim/index.html

National Institute of Standards and Technology. http://math.nist.gov/MatrixMarket/formats.html

Toubia, Olivier and Netzer, Oded. “Idea Generation, Creativity, and Prototypicality.” Columbia

Business School. Copy provided by authors (not published).

Appendix

Most of the code can be found at https://github.com/Centaurific/Blender/

SLURM batch job submission for distributed computing:

```
#!/bin/bash

#SBATCH --time=6:00:00   # walltime
#SBATCH --ntasks=10   # number of processor cores (i.e. tasks)
#SBATCH --nodes=1   # number of nodes
#SBATCH --mem-per-cpu=3G   # memory per CPU core
#SBATCH -J "clustertfidftolsi"   # job name

source activate centaur  # Sets a conda environment

cd '/fslhome/cbell75/Documents/Blender2/wikinetwork'

export PYRO_SERIALIZERS_ACCEPTED=pickle
export PYRO_SERIALIZER=pickle

python -m Pyro4.naming &

srun -n10 --mem-per-cpu=3G python -m gensim.models.lsi_worker &

python -m gensim.models.lsi_dispatcher &

sleep 1m # Ensures that the previous processes are finished

python clustertfidftolsi.py
```

SLURM report on the distributed computing:

```
(centaur)-bash-4.1$ scontrol show job 10133282
JobId=10133282 JobName=clustertfidftolsi
  UserId=cbell75(21962) GroupId=cbell75(22307)
  Priority=107119 Nice=0 Account=dav QOS=normal
  JobState=RUNNING Reason=None Dependency=(null)
  Requeue=0 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
  RunTime=00:47:52 TimeLimit=06:00:00 TimeMin=N/A
  SubmitTime=2016-04-26T17:41:44 EligibleTime=2016-04-26T17:41:44
  StartTime=2016-04-26T17:42:43 EndTime=2016-04-26T23:42:43
  PreemptTime=None SuspendTime=None SecsPreSuspend=0
  Partition=m7 AllocNode:Sid=m8int02:145137
  ReqNodeList=(null) ExcNodeList=(null)
  NodeList=m7-20-9
  BatchHost=m7-20-9
  NumNodes=1 NumCPUs=10 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
  TRES=cpu=10,mem=30720,node=1
  Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
  MinCPUsNode=1 MinMemoryCPU=3G MinTmpDiskNode=0
  Features=(null) Gres=(null) Reservation=(null)
  Shared=OK Contiguous=0 Licenses=(null) Network=(null)
  Command=/bluehome3/cbell75/realcluster.sh
  WorkDir=/bluehome3/cbell75
  StdErr=/bluehome3/cbell75/slurm-10133282.out
  StdIn=/dev/null
  StdOut=/bluehome3/cbell75/slurm-10133282.out
  Power= SICP=0
```

The 20 most prevalent topics in our Wikipedia corpus. The presence of administrative words

and music/sports bots is prevalent, and often shows up when shorter documents are queried

```
(0, '0.720*"jpg" + 0.676*"file" + 0.100*"png" + 0.072*"logo" + 0.039*"cover" +
0.029*"wikipedia" + 0.028*"poster" + 0.027*"delet" + 0.024*"articl" +
0.020*"imag"')
(1, '0.595*"delet" + 0.563*"wikipedia" + 0.517*"articl" + 0.122*"categori" +
0.087*"wikiproject" + 0.067*"spam" + 0.064*"linkreport" + -0.044*"jpg" +
0.039*"log" + 0.036*"sockpuppet"')
(2, '0.932*"categori" + -0.126*"delet" + 0.112*"establish" + -0.086*"wikipedia" +
0.076*"peopl" + 0.057*"album" + 0.053*"footbal" + 0.049*"counti" +
-0.044*"articl" + 0.042*"state"')
(3, '0.352*"refer" + 0.290*"may" + 0.239*"wikiproject" + 0.226*"link" +
0.226*"spam" + 0.209*"linkreport" + -0.207*"categori" + 0.197*"report" +
-0.160*"delet" + 0.133*"record"')
```

(4, '0.538*"refer" + 0.412*"may" + -0.330*"wikiproject" + -0.312*"spam" +
-0.298*"linkreport" + -0.252*"link" + -0.204*"report" + 0.136*"delet" +
0.121*"articl" + -0.114*"wikipedia"')
(5, '-0.465*"refer" + -0.331*"may" + -0.201*"wikiproject" + -0.189*"spam" +
-0.186*"linkreport" + -0.175*"categori" + -0.109*"link" + 0.091*"album" +
-0.090*"wikipedia"+ 0.085*"school"')
(6, '-0.850*"png" + 0.343*"jpg" + -0.333*"logo" + -0.205*"file" + -0.035*"gif" +
-0.032*"svg" + 0.022*"album" + -0.015*"stemma" + -0.014*"jpeg" +
-0.013*"screenshot"')
(7, '-0.511*"link" + -0.354*"report" + 0.329*"wikiproject" + 0.268*"linkreport" +
0.258*"spam" + -0.196*"account" + 0.181*"wikipedia" + -0.166*"involv" +
-0.151*"record" + -0.139*"linkwatch"')
(8, '-0.465*"album" + 0.293*"district" + 0.217*"villag" + 0.203*"popul" +
0.202*"counti" + -0.196*"song" + -0.171*"releas" + -0.161*"band" +
0.146*"census" + 0.125*"link"')
(9, '0.433*"album" + -0.338*"footbal" + -0.242*"leagu" + -0.234*"team" +
-0.219*"season" + -0.164*"play" + 0.164*"district" + 0.163*"song" +
-0.157*"portal" + -0.141*"game"')
(10, '-0.837*"portal" + -0.320*"select" + -0.190*"pictur" + 0.128*"footbal" +
0.103*"leagu" + -0.101*"anniversari" + 0.095*"season" + 0.086*"team" +
0.085*"delet" + -0.081*"articl"')
(11, '0.316*"album" + 0.257*"footbal" + 0.241*"district" + 0.219*"portal" +
-0.208*"utc" + -0.182*"sockpuppet" + -0.178*"film" + 0.175*"villag" +
0.172*"leagu" + 0.153*"popul"')
(12, '-0.675*"sockpuppet" + -0.310*"investig" + -0.270*"wikipedia" +
-0.224*"utc" + 0.169*"linkreport" + 0.155*"articl" + 0.152*"delet" +
-0.141*"suspect" + 0.124*"spam" + 0.122*"wikiproject"')
(13, '0.517*"speci" + 0.479*"genus" + 0.358*"famili" + 0.261*"moth" +
0.180*"beetl" + -0.149*"district" + 0.129*"describ" + 0.124*"cerambycida" +
0.112*"snail" + 0.104*"found"')
(14, '-0.633*"film" + 0.419*"utc" + -0.254*"sockpuppet" + 0.156*"data" +
0.132*"album" + 0.120*"articl" + 0.116*"local" + -0.114*"investig" +
0.106*"entri" + 0.101*"detail"')
(15, '-0.493*"film" + -0.410*"utc" + 0.230*"sockpuppet" + 0.217*"station" +
-0.146*"district" + -0.141*"data" + -0.111*"articl" + 0.106*"school" +
0.105*"investig" + -0.105*"entri"')
(16, '-0.321*"household" + 0.274*"district" + -0.252*"age" + 0.251*"station" +
-0.204*"peopl" + -0.203*"median" + -0.192*"surnam" + -0.170*"femal" +
-0.167*"incom" + 0.146*"villag"')
(17, '-0.893*"logo" + 0.334*"png" + -0.141*"svg" + -0.121*"gif" + 0.072*"file" +
-0.068*"station" + 0.061*"elect" + -0.057*"household" + -0.042*"age" +
-0.040*"film"')
(18, '0.646*"station" + 0.289*"railway" + -0.250*"elect" + -0.165*"school" +
-0.158*"district" + 0.142*"line" + -0.142*"logo" + -0.125*"parti" +
0.109*"household" + 0.103*"film"')
(19, '-0.793*"surnam" + -0.409*"peopl" + -0.214*"notabl" + 0.128*"establish" +
-0.117*"includ" + 0.105*"household" + 0.084*"age" + 0.069*"median" +
0.067*"film" + 0.065*"unit"')