A Theory of Finite Indexing

Clinton P. Mah and Raymond J. D'Amore, formerly of PAR Technology Corporation

14y March 2024

A Half-Century of
N-Gram Statistical Text Analysis

supercalifragilisticexpialidocious

```
super
  perc
    rca
     cali
       lif
        ifr
         frag
           agil
             ili
              list
                stic
                 tice
                   cex
                    exp
                     xpi
                      pia
                       ial
                        ali
                         lid
                          ido
                           doc
                            oci
                             cio
                              ious
```

TABLE OF CONTENTS

1. Introduction

n 1975, Professor Girard Salton of Cornell University published his monograph *A Theory of Indexing*. This became a classic of natural language text analysis, and its ideas are still widely employed in 21st Century information systems.

Much of the knowledge of modern civilization is recorded in natural language text. This is hard for computer systems to access because the structure of any natural language will be complex, highly unpredictable, and often idiomatic. Different languages, or even dialects in the same language, can operate under different rules and can be hard to map into each other.

For example, Mandarin Chinese and English are the two most widely spoken languages in the world today. They evolved independently over many millennia and have made incompatible choices in grammar that make translations between them problematic. Chinese has no true relative clauses, an awkward passive voice, and peculiar sentence particles.

Salton's insight was that messy linguistic details are unimportant in text data applications dealing only with general content. To see what a text document is about, it is often enough just to see what words occur in the document, regardless of how they are formed or how they are grammatically related.

In particular, Salton would represent a document d in a data collection D as a numerical vector v^d defined as

$$v^d = \langle f_w^d \rangle_{w \in W}$$

where W is the set of all distinct root words in D and f_w^d is the number of times that a word or word root $w \in W$ occurs in document d. Index set W will be unbounded, since new documents entering a data collection will generally contain words or word roots never seen before.

Given another vector u^e for another document e in D, we can compute a raw simlarity score between them as an inner product of their vectors

$$IP(u^e, v^d) = \sum_{w \in W} f_w^e \cdot f_w^d$$

This formula works only if documents d and e both are finite.

The inner product here is awkward for measuring similarity. It is sensitive to the length of documents being compared and has no upper bound. Salton tried to skirt these problems by normalizing document vectors to a unity length in a vector space assumed to be Euclidean.

So, instead of working with a vector v^d, Salton used

$$\hat{v}^d = \frac{v^d}{[\ \sum_{w \in W} (f_w^d)^2\ ]^{1/2}}$$

and then defined

$$cosine(u^e, v^d) = IP(\hat{u}^e, \hat{v}^d)$$

This is called the cosine measure of similarity. It can be interpreted as the cosine of the angle between $\hat{u}$ and $\hat{v}$ in an infinite-dimensional Euclidean space of document vectors. We have

$$0 \leq cosine(u^e, v^d) \leq 1$$

where 1 is an exact match and 0 is none at all. The cosine measure is often used in finding the document d in a collection D best matching a query q and also in computing pairwise similarity of documents for clustering them. These procedures benefit from some minimum threshold on similarity to reduce computation; but it is unclear when a given cosine measure is high enough to be significant.

2. Finite Models

A standard engineering practice is to define simplified models to expedite the design for a complicated system. For example, we can reduce a bridge to a structural skeleton on which to check the stresses and strains that it must handle under expected use. Other models can also help in finding possible hot spots on a packed silicon chip at various voltage of operation or speeding realistic computer animation of long hair blowing in the wind.

Salton's numerical vector representation of text documents is such a model. It has had success in many information applications, but is only one possible approximation of reality. Its initial assumptions do show definite shortcomings. For example, the words employed for indexing will generally be non-orthgonal in meaning, contrary to any Euclidean interpretation. Infinite dimensionality itself will also raise practical problems. Can we do better?

For a half century, unbounded vector dimensionality has been widely acceptable for indexing. Fixed, finite dimensionality, however, allows for simpler modeling and implementation and will have theoretic advantages. It will also facilitate the export of indexed data for further analysis. In particular, it offers a convenient way to feed data into deep neural nets, the signature technology in the 21st Century for artificial intelligence.

The input of a neural net must be a numerical vector of fixed, finite dimensionality, corresponding to the number of nodes in a net's input layer. We can adjust this number, but that then results in a new net to retrain in all its layers. Some net retraining will be needed even with vector input of fixed dimensionality M, but this should be less often and should preseve more prior training.

To transform a vector counting finitely many occurrences of words or word roots $w \in W$ to a new vector counting occurrences of $x \in X$, a finite index set, we have many options. The simplest is a linear transformation

$$T : \{v^N\} \rightarrow \{v^M\}$$

where N is the current size of W, v^N is a vector in a space indexed by W, with finitely many non-zero entries, and v^M is a vector in a space indexed by X.

T might be represented as a rectangular matrix with N rows of length M, where N will keep growing. Implementing such dynamic matrix will call for automatically generating a new row for each new $w \in W$. Vector v^N will be transformed into a finite indexing vector in X by multiplying it as a column vector with matrix T.

$$T(v^N) = [T] \circ v^N$$

Since there are infinitely many ways to define T, it should be easy to map an infinite Salton indexinghyperspace into a finite one. The challenge is to get a finite index set suitable for gauging vector similarity and other attributes.

3. Evaluating Finite Index Sets

Our discussion now will shift exclusively to English text written in a Latin alphabet. In non-alphabetic languages like Chinese (ideographic), Korean (syllabic), or Japanese (ideographic plus two different syllabaries), defining a finite set of indexing features for text will be different and much harder.

Let us first lay out some general guidelines for setting up finite indexing for English. We shall mainly look at the likelihood that each x in a candidate index set X will occur in a document d in D. With such probabilities $\{ p_x \}$, along with some basic information science and knowledge of English. We then establish four general goals to aim for.

power Text in widely used languages like English and Chinese often assume that a reader will understand about only about 10^4 distinct words. In America, this is about the vocabulary of the average educated sixth grader, which makes 10^4 a reasonable minimum goal for the size of a finite index set X. That should be large enough to rival our sixth grader in distinguishing between different kinds of possible content.

coverage If Salton's vectors distinguish document content well, then indexing with a finite X mapped alternative needs to remain fairly close to indexing with W. We can accomplish this in various ways, but a simple one here is to require that for any word-indexed vector v^d

$$v^N \neq 0 \;\rightarrow\; T(v^N) \neq 0^M$$

where 0 is the zero vector of Salton's W index space, and 0^M is the zero vector for the finite-dimensional space indexed by X. Note that the implication arrow $\rightarrow$ goes only one way. This reflects that mapping of infinite to finite vectors will introduce irreversible indexing noise. Our goal is to control such noise so that vectors in X still are helpful for computing numerical similarity measures.

entropy The frequency of a set of elements in any natural language text tends to follow a statistical power law. This typically means that that a small subset of elements of any X or W will account for most of their occurrences in any data collection. This imbalance leads to inefficiency in indexing, but cannot be completely eliminated. The problem is usually called Zipf's Law and is often expressed mathematically as

$$F_x \propto \frac{1}{rank\ x}$$

where F_x is the total frequency of feature $x \in X$ in a text data, and $rank\, x$ is the ordinal ranking by frequency of x versus all other features in X. (1 = highest, 2 = next highest, and so forth). Zipf's Law seems a bit strange, but it helps to make natural languages more learnable in a home or other informal setting. The imbalance in indexing is often measured by Shannon's measure of information entropy for an index set X.

$$H(X) = -\sum_{x \in X} p_x \cdot log_2 p_x$$

where p_x is the probability of index feature x in a data collection D. A perfectly balanced set of indexing features X would have $p_x = p_y$ for all x and y in X, with $p_x = p_y = 1/M$. In this case, we have

$$-\sum_{x \in X} p_x \cdot log_2 p_x = -log_2 \frac{1}{M} \cdot \sum_{x \in X} p_x = log_2 M \cdot 1 = log_2 M$$

which is maximum information entropy for M indices.

$$\max H(X) = log_2 M$$

The presence of the log_2 means that entropy will be expressed as bits of information. Minimum entropy is when $p_x = 1$ for some single x.

`

$$\min H(X) = p_x \cdot log_2 p_x = 0$$

or zero bits of information.

Maximum entropy is unachievable because of Zipf's Law, but we always want to make entropy as large as possible.

independence

This is a weakness of indexing text with words W. In English, words can be variously synonymous; for example, RED, RUFOUS, SCARLET, CRIMSON, RUBY, CARNELIAN, or CARMINE. We can ignore this overlap sometimes, but with a finite X, every $x \in X$ must pull its own weight. Otherwise, the indexing effectiveness of X will be illusory.

Independence will be useful theoretically. A pair $x, y \in X$ will be independent when

$$p_x \cdot p_y = p_{xy} \quad for \quad x \neq y$$

where p_{xy} = the probability of x and y both occurring in the same document. This condition is costly to verify. For $M = 10^4$, we must compute p_{xy} for $(M - 1)(M / 2)$ instances. Furthermore, since our estimates of probabilities are based on text samples, they will be less reliable for rarer indices. So, an independence test should be a looser inequality

$$|p_x \cdot p_y - p_{xy}| < \epsilon$$

for fixed small $\epsilon > 0$. With Zipf's Law, most probabilities p_x, p_y, and p_{xy} for $x, y \in X$ will be small for large M. It is then likely that $p_x \approx p_y \approx p_{xy} \approx 0$, which means that some independence is already given in a large X. To get more, we can just exclude some high probability n-gram indices from multinomial calculations. This may seem convoluted to get more independence of $x \in X$, but the mathematics seem workable.

We are close. Must we really satisfy our inequality above for all p_x? Can we live with a few exceptions and work with incomplete independence? Can we get by just with good enough? Maybe we scan vectors somehow mask out x, for which p_x is unusually high? That could greatly help our statistical analyses. So, let us try out various X with actual text data and see what happens.

We now have all the basics for finite text indexing. Let us define an actual finite set X of text indexing features for English and to estimate the probabilities of each feature.

4. Index Feature Probabilities

Let us be explicit on p_x. Finite indexing lets us estimate such probabilities easily. We only have to sum the occurrences for each $x \in X$ over a large, though finite, sample D of text data. The estimated p_x for finite X will simply be

$$p_x \approx \frac{\sum_{d \in D} f_x^d}{\sum_{d \in D} \sum_{x \in X} f_x^d}$$

This formula works only when X and D are both finite. The quality of estimates will of course depend on the collection D, but with a large enough D, the numbers should be fairly close to those for English in general.

Probabilities for a set X with about 10^4 features can be computed quickly. When data collections are dynamic, however, we have to update probabilities often. Our current strategy is to use all of D as our sample data for estimates. We can index new text data in small batches and get revised probabilities after each batch.

In Salton's system, a word index set W can grow without bounds, though at any time, it will be finite. One might try to estimate probabilities for $w \in W$ as above, but the arrays for keeping partial sums will grow beyond any fixed bound. Also, most words in W will be rare, resulting in poor probability estimates. One typically wants at least 10 to 20 occurrences of any indexing features in D for good results. This is easier to achieve with finite indexing.

Salton's system avoids working with probabilities. Functions of the counts of words, such inverse document frequency, are used to weight query terms; but they are only peripheral to system operation. Finite indexing, in contrast, will be built entirely around probabilities. Even with estimation error, indexing probabilities will let us assess the importance of individual text items better and to get a handle on the significance of similarity scores for a pair of items.

The formula for p_x above might seem too obvious to be worth mentioning, but it cannot be computed for an infinite index set like W. The ability to estimate probabilities is just our first dividend from finite indexing. They will later allow more rigorous analyses of a data ollection. In the meantime, here are some actual probabiities computed for a data set.

```
probability range= 0.000011 : 0.004740
6488 non-zero indices
based on 87755 total occurrences of indices

indices with highest probabilities of occurrence

(  341):   0.004740
(11786):   0.004410
(11354):   0.004387
(  318):   0.003567
(12702):   0.003487
(11998):   0.003339
(  343):   0.003031
(  369):   0.002815
(11706):   0.002723
(10445):   0.002484
( 6467):   0.002427
(  327):   0.002336
( 3035):   0.002302
( 2739):   0.002165
( 7021):   0.002051
(   48):   0.001926
( 4259):   0.001914
( 3396):   0.001892
(  313):   0.001880
(12368):   0.001857
(    1):   0.001778
( 2964):   0.001766
( 6172):   0.001732
(   45):   0.001709
(10746):   0.001698
( 5999):   0.001675
(  167):   0.001630
(10959):   0.001595
( 9395):   0.001584
( 2375):   0.001561
(  334):   0.001504
(11981):   0.001481
… (omitted)
(11799):   0.000991
(  180):   0.000980
(10209):   0.000980
(10929):   0.000980
( 5262):   0.000969
(11179):   0.000969
(11525):   0.000969
(11535):   0.000969

total percentage of occurrences for top 100 = 15.46
```

The number in parentheses in a numerical code for a finite index, here ranging from 1 to 13800. The second column is a probability in descending order, giving an idea of Zipf's law at work in actual data. The top 100 indices account for over 15 percent of all index occurrences. The minimum non-zero probability for an index was 0.000011.

5. Finite Indexing, Step by Step

The four requirements of Section 3 above for a finite index set X mean that

$$X \not\subset W$$

since we would otherwise violate the coverage requirement. It is possible, however, to put a few whole words into a finite X. With text data in English, a better general indexing alternative is to count mostly word fragments instead of full words. This will meet our coverage requirement, though we still much address our power, information, and independence requirements. A word fragment approach will have important consequences.

Consider fragments of single letter (1-grams[1]). Anyone familiar with cryptanalysis in English will know E, T, A, O, N, I, S, H, R, D, L, and U, in descending order of expected frequency in English text; these will be the most common letters in almost any nontrivial collection of English text data, and their statistics are mostly unhelpful for telling documents apart. They do satisfy our coverage requirement though, which is a start.

How about alphanumeric 2-grams? English has 36×36 = 1,296 of them. This is getting closer to our target of 10^4, but English cryptanalysis tables indicate that certain alphabetic 2-grams like TH and AN still have notably high frequency, lowering the information entropy of indexing; independence also remains an issue. Here is how we might count 2-grams in a bit of text.

```
resource
re
 es
  so
   ou
    ur
     rc
      ce
```

This satisfies our coverage requirement and is better than indexing with just 1-grams, but documents can still be hard to tell apart. We need a larger X to work with.

We get more indexing features with alphanumeric 3-grams, but will run into a new problem. In English, there will be 36×36×36 = 46,656 alphanumeric 3-grams. This seems out of control, since most will be nonsense like JRJ, HHQ, EIE, or ZKM, There is no reason to have an index set X so bloated. So, why not just continue to index with all 2-grams to meet our coverage requirement, but add on selected alphabetic 3-grams to increase the power of our index set?

The idea is to count a selected 3-gram whenever possible and fall back on 2-grams only when necessary. For example, if we introduce SOU for indexing, our analysis above becomes

```
resource
re
 es
  sou
    ur
     rc
      ce
```

[1] Google takes “n-gram” to mean a sequence of n words. Wee take it to mean a sequence of n letters or digits. This is the practice of cryptanalysis, which long predates Google.

Such counting of n-grams of different length will be called “the rule of non-redundant indexing.” If a segment of text is already covered by a longer indexing n-gram, we do not count any shorter n-grams falling entirely within that segment

Adding just a single frequent 3-gram to our X will raise indexing entropy. It can also increase independence overall. With only 2-grams for indexing English text, the probabilities of SO and OU in a document will show dependence. If adjacent occurrences of SO and OU are counted as a single occurrence of SOU, our non-redundancy rule will mask some of that dependency. This will make SO and OU more useful as individual index features.

In general, the addition of a single frequent (n+1)-gram to an n-gram index set will help advantageous. The gain here will be cumulative for further additions, which suggests how we might reach a goal of around 10^4 indexing features. Zipf’s Law will help us here: we shall only have to look at a few frequent n-gram indices to find (n+1)-grams that should make a big difference in indexing.

Selecting a suitable subset of alphabetic 3-grams can be complicated. For expediency, we started with 3-grams formed by appending a single letter to the K most frequent alphabetic 2-grams in English text. The choice of K = 216 was because we wanted 3-grams of the form QU■ in our index set X. We then get 216 × 26 = 5,616 alphabetic 3-grams, giving us a little under 7,000 2- and 3-gram indices, good enough for initial experiments.

6. Some Indexing N-Grams

We have been circling n-grams from a distance. Efficient software calls for plunging into the nuts and bolts of indexing, however, finite or otherwise. We then added longer n-grams more selectively to get to our goal of 10^4 indices. Here are actual examples of current n-gram indexing for English text, listed by type:

Alphanumeric 2-grams:

```
00 01 02 03 04 ... zv zw zx zy zz
```

Alphabetic 3-grams:

```
aba abb abc abd abe ... yev yew yex yey yez
```

Alphabetic 4-grams:

```
allí bold cree fair lace nsel oddl rose spac word
```

Alphabetic 5-grams:

```
center draft eight forge honey mater prote screw thres worth
```

User-defined n-grams

```
-000000 -control -person -structure associate- cyber- fbi- opec-
```

An n-gram index reference can be stored as a single 2-byte short integer code. This will simplify the data structures for text analysis.

The type of an n-gram can be inferred by the range in which the code falls. These ranges currently are: user-defined (1 - 2000), 2-gram (2001 - 3350), 3-gram (3351 - 9950), 4-gram (9951 - 12050), 5-gram (12051 - 13050). Empty space for expansion is still available in every range of indexing.

These five built-n n-gram classes have different histories. The 2-grams can cover every text extent of two or more letters or digits. The 3-grams divide the probability of frequent 2-grams among multiple 3-grams to improve indexing entropy. The 4- and 5-grams evolved over time, selected for occurring in multiple words and common names. This has taken a long time, but has improved the coherence of cluster and reduced overall noise.

Later sections will explain index selection in more detail, but for now, we just need to acknowledge that our final destination might seem strange. A skeptic might only see finite indexing as fizzling out as a competive information processing option. So far, however, our finite n-gram indexing has worked out well in every test.

These are in no way optimal recipe for indexing English text. It may be enough, though, just to have many n-gram indices likely to occur in a data set.

7. A Theoretical Keystone

At some point, we have to think about how to build practical software to achieve important information goals. First, we must run experiments to test our ideas objectively and rigorously. The results of such work will be presented in subsequent sections, but we begin by providing a more complete framework for evaluation.

Although we have nibbled around the edges of what an index set X might be, we have yet to show any advantage for finite indexing. So, let us get this detail out of the way, so as to gain deeper understanding of all text indexing, finite or otherwise. This has no parallel in Salton's formulation of text indexing.

If we model a collection of documents D as points in a finite M-dimensional numerical vector space, we like Salton can define an inner product of a pair of vectors as a raw measure of the similarity of their corresponding documents. The finiteness of the vectors, however, will also allow us to estimate the distribution of raw inner products expected by chance, given the estimated probabilities $\{ p_x \}$ for $x \in X$. This allows us to use a multinomial model for inner products computed for vectors with dimensions corresponding to X. A precondition for a multinomial model is that each $x, x' \in X$ will have a frequency of occurrence across documents that has a binomial distribution, and x must occur independently of x' when $x \neq x'$. This is hard to meet directly; for example, certain words tend to clump in fewer documents than a binomial assumption would predict. The problem is less with n-gram indexing in that any n-gram can occur across many different words.

With a bit of manipulation of random variables, a multinomial distribution is a good model for the AW inner products between finite vectors v and v'. Its most important statistics will be

$$E[s(v, v')] = LL' \cdot \left[\sum_{x \in X} a_x p_x^2 \right]$$

$$Var[s(v, v')] = LL' \cdot \left[\sum_{x \in X} a_x p_x^2] \right] + \ (L + L' - 2) \cdot \left[\sum_{x \in X} a_x p_x^3 \right] \ -$$

$$(L + L' - 1) \cdot \left(\left[\sum_{x \in X} a_x p_x^4 \right] \ - \ \left[\sum_{x' \in X \ \wedge \ x' \neq x} a_x a_{x'} p_x^2 p_{x'}^2 \right] \right)$$

where

$$L = \sum_{x \in X} f_x^d, \quad L' = \sum_{x \in X} f_x^{d'}$$

$$a_x = 1 \ (by \ default)$$

The full derivation of the expected value and variance of the raw inner product is in the Appendix. This assumes that indices in X are independent. As noted above, independence is hard to verify, but we can compute the distributions of actual inner products for random pairs of vectors from our finite space in X and compare them to theoretical predictions to see if vector dimensions are independent enough.

If so, then we then can compute a statistically scaled similarity

$$sim(v^d, v^{d'}) = \frac{IP(v^d, v^{d'}) - E[IP(v^d, v^{d'})]}{[Var[IP(v^d, v^{d'}]^{1/2}}$$

This similarity score will be in units of standard deviations, easier to interpret than Salton's hyperspace cosines. The multinomial noise distribution will be unimodal, but we need not think of it as approximately Gaussian. When separation of signal versus noise is high, it is good enough to report scaled similarity scores just in standard deviations without worrying about their actual statistical p values.

8. Unpacking Scaled Similarity

Statistical scaling lets us look at indexing in ways that expand our thinking about what text information systems can do. Interpretability of similarity can be quite valuable, even with only an approximate model of text. Extreme cases can be instructive. In finite indexing, a problem arises with identical or nearly identical documents, which would have a maxinmum cosine similarity score ≈ 1.0.

Scaled similarity has no upper bound. The score between an identical pair of documents, will depend inversely on their length and more subtly on the probabiities of a reference text data set. Ducument length dominates here because longer documents will typically have noise distributions with a larger expected value and variance. From the prevous section:

$$E[IP(d, d')] = LL' \cdot S_2$$

$$Var[IP(d, d')] = LL' \cdot [S_2 + \ (L + L' - 2) \cdot S_3 - (L + L' - 1) \cdot [S_4 \ - \ S_{22}] \ ]$$

.where the $S_{\blacksquare}$ are precomputed sums dependent only on $\{ p_x \}$. For example,

$$S_2 = 0.00293167$$

$$S_3 = 0.00000200$$

$$S_4 = 0.00000000$$

$$S_{22} = 0.00000174$$

These sums depend only on the overall probabilities of a data set. They can be computed ahead of time and used for statistical scaling just by suppying the L and L' for a document pair. Given $L = L'$ = 1, raw similarity r = 1 when documents are identical.

The expected value for any r is S_2 with a variance of $S_2 + S_{22}$. This will be true for any pair of documents with $L = L'$ = 1. With two identical documents, we get

$$E[r] = 1 \cdot [0.00293167]$$
$$Var[r] = 1 \cdot [0.00293167 + 0.00000174] = 0.00293341$$

Everything now is arithmetic:

$$sim = \frac{1 - 0.00293167}{\sqrt{0.00293167 + 0.00000174}} = \frac{0.00706833}{\sqrt{0.00293341}} = 755.88\ \sigma$$

This is an astounding significance, but such numbers could be expected for minimally short documents. We therefore should be extra careful about finite indexing of extreme cases.

Now suppose $L = L' = 2$, with two identical documents, both with only one distinct index.

Then have r = 4 and

$$E[r] = 0.01172668$$
$$Var[r] = 4 \cdot [0.00003688 + 2 \cdot 0.00000200 + 3 \cdot 0.00000174]$$
$$= 0.00003688$$

Statistical scaling then gives us

$$sim = \frac{4 - 4 \cdot 0.00293167}{\sqrt{0.00003688}} = 656.73\ \sigma$$

This is still astounding, but not as much as with minimally short documents. The match scores are definitely different for $L = L' = 2$ intead of 1; the scores in Salton's system for both cases be 1.0. We shall see lower scaled significance scores for matching longer documents.

9. Evaluation Framework

We now have everything necessary for some rigorous testing of finite indexing counting 2-and 3-grams as features in text. This X with about 7,000 indices is shy of our goal of about 10^4, but should be adequate to test our ideas about finite indexing before going to the next stage of R&D, requiring extensive writing of actual software.

Salton evaluated the performance of document search with word indexing by doing it on a standard collection of documents D with predefined queries Q. Human judges had to examine each document $d \in D$ to determine which were relevant to each query $q \in Q$. One could then run every q to get its top K matches. The number of relevant matches *r* for a query would then let us compute its "recall" as *r* / (total number of documents in D relevant to the queries) and its "precision" as *r* / K.

For finite indexing, we could do the same, but our multinomial model of raw similarity allows for a simpler approach. We can eliminate the need for subjective judgments of relevance as well as support more extensive testing of a system. We can do this quickly for new data collections of text data of interest to current information system users. The key here is finite indexing and the multinomial model for raw similarity scores for document pairs. It begins by getting a large representative sample D' of text documents.

- For $d' \in D'$ and some fixed $L \geq 100$, obtain a segment of text that will have an index vector $v_A^{d'}$, with sum of counts = L.

- For $d' \in D'$, obtain a different text segment to get an index vector $v_B^{d'}$ with sum of counts = L.
- Compute raw inner products for random pairs of index vectors $v_A^{d'}$. This will provide an actual noise distribution to compare against multinomial predictions.
- Compute raw inner products for all corresponding vectors $v_A^{d'}$ and $v_B^{d'}$. This will be an actual signal distribution.
- Determine its separation of the signal distribution from the noise distribution.

That is all. Any successful finite indexing must have a noise distribution closely modeled with a multinomial assumption and also show good separation between signal and noise distributions. The whole objective evaluation process wsill take a few weeks from scratch with any new data set. No human judges are needed.

10. Text Preprocessing

The next section will report actual results from the initial finite indexing experiments. To interpret these results properly, one should be aware that our indexing code incorporated some legacy text processing tools. These are familiar to most people working in text data indexing, but they do greatly affect the statistics in n-gram finite indexing and so is part of how we select n-gram indices.

The first tool relates to grammatical function words in Indo-European languages, including English. These help to organize sentences and can account for a third or more of word occurrences in English text; for example, THE, AND, WITH, BY, FOR, IT, WHAT, NEVER. They typically carry no denotative meaning by themselves, and most indexes will omit (“stop”) them. There is no consensus, however, on which words to stop.

Some purists in machine learning would have no stop list at all, holding that automata like artificial neural nets are capable of figuring everything out themselves. On the other hand, this requires much more complexity in an automaton, which will only slow down an experimental schedule. Machine learning after all is not what we want to evaluate. Our choice was a maximal chosen stopword list, so that n-grams like THE and AND could be more useful as indices.

A second tool was needed for word suffixes. English has two kinds: inflectional and morphological. English regular inflectional suffixes are -S, -ED, and -ING at the ends of nouns and verbs. They appear everywhere in English text data and can produce n-grams that make documents of different content seem more alike in their indexing vectors.

Common English morphological suffixes like -MENT, -SHIP, -OR, IZE are well known. These usually serve to change the part of speech for a root; for example, the noun IDEAL becomes the verb IDEALIZE. Since parts of speech are a finer distinction than needed in a text index for computer information systems, morphological suffixes tend to be removed. This was maximal in the experiments for evaluating the effectiveness and utility of n-gram indexing.

Proper removal of word suffixes (“stemming”) is much more complicated than stopword removal. The problem is that proper detachment of suffixes requires knowledge of English spelling rules. For example, MATING becomes MATE, while MATTING becomes MAT. Such details seem to be a burdensome chore, but can improve the quality of indexing. Fortunately, good stemmers are readily available for download.

Inflectional ending like -S, -ED, and -ING are harder to remove than morphological suffixes. Inflections are much more frequent, and have more special cases to recognize in rules. This effort is necessary, however, because we already have plenty to contend with. Inflectional stemming makes n-grams like TED and ING more helpful for indexing. Inflectional stemmers are widely available, although they will differ in breadth and accuracy.

A third preprocessing decision related to how one should count occurrences of n-gram indices in a document. John Tukey in his own classic book *Exploratory Data Analysis* suggests that, in any statistical analysis of text data, one should replace raw counts with lower transformed counts to lessen the effect of Zipf's Law; for f_x^d in document d, Tukey suggests $\log_2(f_x^d + 1)$ in a document vector.

Testing of n-gram indexing showed a transform does improve performance by lessening the impact of high-frequency indices in X. It reduces the variance of our model noise distribution. There seems to be no obvious best transform, but we have settled on a rounded integer approximation of logaithms of base 2, which seems good enough.

11. Objective Performance Testing

The idea of finite indexing of text data emerged shortly after Salton's original monograph was published. Many information users were dissatisfied with the whole-word indexing systems then being fielded by the U.S. military. This was before the IBM Personal Computer and before the Internet. Hardware was extremely limited by small hard drives and random-access memory. Information engineers had to cram code into small spaces.

All alphanumeric 2-grams and a 3-gram subset were in our first finite indexing test. The 3-grams were inefficient, including many low-value 3-grams like THZ, ANB, and EEJ; but we wanted something quick for basic experiments to show the practicality of n-gram indexing; optimization could wait.

We had two main testing objectives:

- Validate a multinomial model of raw similarity.
- Show discernible separation between signal and noise in raw similarity.

For our data sets, we took whatever was readily available before the Internet:

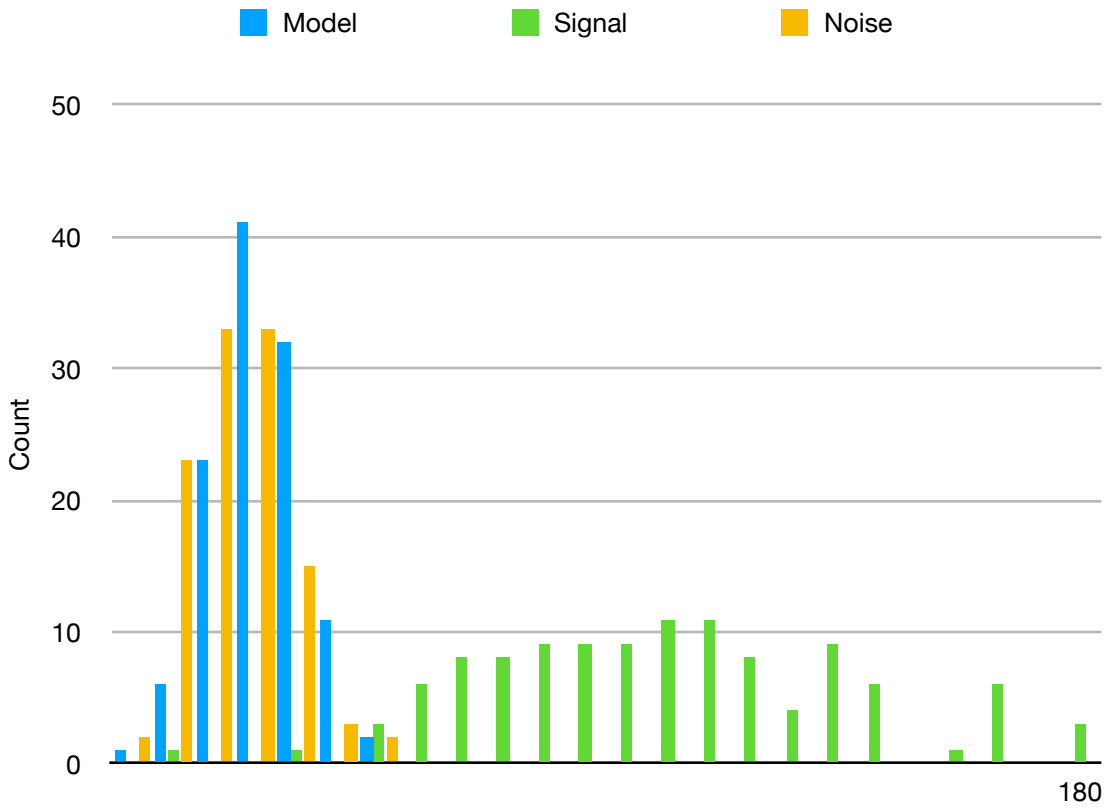
- Newswire stories collected for a previous project by a prospective sponsor (= News),
- Essays from the Norton Anthology, retyped (= Literary).
- A file of abstracts of technical papers in solid-state physics and electronic circuit design; these had been saved on Hollerith punchcards (= Science).

The data sets were indexed separately with our X of about 7,000 2- and 3-grams. It was unclear how well finite indexing would work with diverse data. After 40 years, many things can get lost in changes of employment along with interstate moving: the original software, hardware, data, and raw results. What survives is a summary of results from an unpublished technical paper written shortly after the experimental results were first available. The essential findings are as follows:

Data Set	Noise μ	Noise σ	Model μ	Model σ	Signal μ	Signal σ
News	13.7	6.2	14.7	4.2	34.1	10.6
Literary	12.9	4.6	13.6	4.0	25.6	11.5
Science	19.6	7.9	20.3	5.2	44.0	12.7

This indicates that finite n-gram indexing is succeeding. We have reasonable matches between our actual noise distributions and multinomial predictions, although the model variance underestimates actual noise variances.

We can get a more dramatic view of these results by putting them into a binned histogram with



Raw Similarity Scores: 2-, 3-grams with L = 100 for News

the raw inner product scores along the X axis and counts along the Y axis. These plots have become iconic for finite indexing and standard for introducing finite indexing to new audiences.

This chart is just for the News data collection, showing a good fit of its noise distribution with the a multinomial model and good separation of signal and noise. The shape of plots for Literary and Science looked similar, though with less separation of signal and noise. The graphs show plainly what a table of means and variance could not.

Although the multinomial model was an imperfect predictor in our experiments, the obvious separation of signal and noise in all three of our data sets suggested that finite indexing might be good enough already with only 2- and 3-grams; we could start putting together a demonstration application processing indexed only by those n-grams.

12. Scaling Document Similarity

The multinomial is an approximate model of raw document similarity scores, providing a consistent and practical way to map out a document collection. Here is a graph showing separation between noise and signal raw scores. Model performance should improve with eventual expansion of our index set X to 10^4 n-grams. The immediate priority, however, was to show how such a mapping might serve current text data analysis.

Making sense of the variance was a first step. Scientists, engineers, and educators are familiar with the interpretation of standard deviations, which are the square-roots of variance. It is often used to estimate the probability of some measurement happening by chance relative to an error distribution around some mean value of observed measurements. That distribution is typically Gaussian, popularly known as a bell curve.

There are published tables of significance for particular Gaussian standard deviations. For example, a measurement that is three standard deviations from an expected valuatiom has a probability of p=.05. A multinomial distribution has one “hump” like a Gaussian distribution, but will be asymmetric with a long tail to the right, unlike the Gaussian. A Gaussian interpretation is nevertheless not unreasonable.

Because of the Law of Large Numbers, a multinomial distribution for inner products of vectors will approach a Gaussian one as dimensionality M increases. So for $M \approx 10^4$, a Gaussian interpretation of standard deviations for inner-product similarity is reasonable. The catch here is our multinomial underestimation of actual variance, but sufficient separation of signal noise should allow some tolerance of that. Here are the numbers for our three data sets.

Data Set	Signal μ - Model μ	Model σ	Comments
News	1.0	7.0	the best
Literary	0.7	3.0	should be better with bigger L or M
Science	0.7	4.5	

13. Modeling Profile Similarity and Scaling

The development of the multinomial model for the raw similarity of random pairs of document vectors is the cornerstone of finite indexing. It can be only a single brick, however, in the design of practical software. We must able to show how a system with finite indexing will look to an actual user and how it differs from having just another search engine.

A full demonstration of finite indexing still needs to provide the capability of comparing documents in a collection D against a fixed reference profile q, which might serve as a query.

$$q = < q_x >_{x \in X}$$

where the $\{ q_x \}$ are arbitrary real values. This will make q a numerical vector of M dimensions, but this is unlikely to resemble a vector for any document in any collection D. To define a new scaled similarity measure for a document d and reference profile q, we have many options. With finite indexing, this should be simple enough to make modeling its noise distribution easier. We start with

$$Raw(d, q) = \sum_{x \in X} f_x^d \cdot q_x$$

If we rename q_x as a_x and reverse the order of multiplication in the summation, we get

$$Raw(q, d) = \sum_{x \in X} a_x f_x^d$$

which looks almost like raw inner product similarity for a pair of documents, but involves fewer statistical degrees of freedom. This is not a case of a multinomial model applied to an inner

product. Yet, If we do the same mathematics to that in computing expected value and variance for pairwise similarity, we get

was

These noise distribution parameters must be recomputed for each new q; for scaling document for pairwise similarity, we still also recompute them whenever the probabilities $\{ p_x \}$ change significantly.

The statistically scaled document similarity to q will then be

$$sim(d,q) = \frac{(Raw(d,q) - E(Raw(d,q))}{[Var(Raw(d,q)]^{1/2}}$$

With our two types of statistical scaling, we can envision a real-time text information mapping system quite beyond what conventional engines can support. In particular, this system can identify major areas of content in a data stream and also construct its own standing queries for deetecting them. The system always operates under the control of its users, but can run productively for extended periods of time without any intervention.

14. A Context for a Demonstration

We now have two distinct, though similar, ways to calculate a statistically scaled measure of similarity. This is unusual for a text-based information system. It raises the question of how to put such a dual capability into an actual system implementation and how it might extend what we have already in conventional search engines. Many text information users might seem well-served by software widely available on any desktop.

Some textprocessing, however, will have more demanding requirements. Consider a command center gathering streams of data in English from manysources arriving day and night at unpredictable times. The incoming information is often highly perishable and might lead to dire consequences if not identified quickly. So, teams of personnel in shifts keep close watch 24 hours per day seven days per week, even though data streams are quiet most of the time.

In the not-so-distant past, the command center would have an array of teletype machines clattering away, and those on watch would look at the fanfold output and tear off each individual document. If obviously important, it will be immediately passed up the command chain. Otherwise, they will be sorted out by subject and snapped onto appropriate clipboards hanging on a wall for later review. A status board will show what subjects are trending.

Such an operation is chaotic. No one person will see all related incoming information. Some key link can slip through, when data arriving from different sources on different shifts to be pieced togther. In modern command centers, everything is digital and stored on networked hard drives, but its monitoring problems remain. Technology will ease the job of watch personnel in some ways, but can also increase the total amount of data to sift through.

15. Finite Indexing at Work

A system with finite Indexing will never replace search engines, which have proven utility. If someone wants to find text documents cotaining certain words, there is no reason to fragment those words and look for documents containing those fragments. This will only add noise into search results. Command centers, however, need tight control over the matching process and could use more support than a search engine usually provides.

Normal search engines operate interactively. For example, Google allows users to enter a query and get immediate ranked results. A searcher has no guarantee, though, that the top retrieved document will be relevant to the query. Someone has to read at least a description ot top

results. This is acceptable where a searched collection is largely static. It gets more complicated when live data streams carry documents needing to be identified quickly.

A standing query should check every document as it arrives and notify a user right away about a strong match. This can be hard to do. With Boolean logic queries, setup and tuning will call for expert coding and debugging skills. System with numerical similarity scores have an advantage here, but one then has to set the right lower score thresholds for good performance. With the cosine measure, thresholds probably have to be different for each query.

With finite indexing, similarity thresholds will be easier to control. For example, matches assignment profile below 4σ can be immediately forgotten; matches between 4σ and 6σ will be recorded and sorted, but not reported unless specifically requested; matches between 6σ and 8σ will alerts might also be triggered when unreviewed matches above 6σ exceed some limit.

A similar situation holds for link thresholds for automatic clustering even with a different way of scaling similarity. Since any link below 4σ will be too noisy and could produce spurious clusters; it will require trial and error to find a better threshold to produce the number and type of clusters wanted; but around 6σ seems to work out well. A normalized, but unscaled, cosine measure will be much harder to adjust in such situations.

16. Organizing a Demonstration

With only theoretical validation so far, it was premature to start building a complete prototype command center system. Potential users always want to see and understand any new technology before accepting it. For finite n-gram indexing, this would mean actually processing real text data on hardware familiar to them to give an idea of its complexity of operation, the overhead in its conputations, and the quality of results.

For this purpose, an application was built to focus on automatic clustering. This allows one to generate timely maps of the current contents of a dynamic data collection, helpful to most organizations; and it would emphasize what finite indexing could do beyond conventional search engines or even beyond familiar clustering techniques. Our demonstration application would do two-stage clustering to allow for overlapping sets of documents.

The demonstration would execute a series of modules, each writing out intermediate files to be read by subsequent modules and available for later inspection. These modules are as follows:

(a) Divide a text file into document segments, using string patterns to decide where to cut.

(b) Generate one or more finite n-gram vectors for each segment.

(c) Update estimates of n-gram probabilities for a document collection.

(d) Choose a subset of document segments to cluster.

(e) Precompute multinomial noise model statistics needed for the pairwise similarity, and gather vectors of selected segments into a single compact file for faster computation.

(f) Compute scaled pairwise similarity scores for each unique segment pair and make a link when scaled similarity is above a minimum threshold.

(g) Apply a variation of the minimal spanning tree algorithm to get raw clusters, constraining the maximum size of a raw cluster and requiring a minimum density of linkage within each raw cluster.

(h) Produce an n-gram profile summarizing each raw cluster, which will be used to assign items to final clusters. We need to compute a noise distribution for each profile, which will employ our query version of the multinomial model.

(i) Assign document segments to final clusters by comparing their document vectors to the profiles obtained from cluster seeds; those vectors with scaled similarity above a minimum threshold for any cluster profile will be assigned to that cluster.

(j) Summarize each final cluster by finding words or word roots in document segments of the cluster that contain highly weighted n-grams in its profile.

The most computationally intensive step above is (f). To make this faster, we can compare only between text segments in different documents, calculate inner products only for projections of index vectors, and organize data for more efficient pairwise processing. Finite indexing will be a big help here.

In steps (e) and (g), the demonstration will test our two ways of scaling similarity scores. Pprospective users can judge for themselves whether the results of the final step (j) make any sense. With a small of under a thousand documents, one can rerun the demonstration with different processing parameters to check the stability of computation.s

Step (h) will allow a document segment to be in multiple clusters. This is appropriate because a segment of text can relate to multiple subjects.

17. Some Clustering Results

After debugging, the clustering application ran on a batch of newswire stories more recent than that in validation testing. This was done many times to determine how to set processing parameters to get the best clusters to show in a demonstration: the final choice had a minimum of 8σ for pairwise links taken in the formation of raw clusters and a minimum of 6σ for a profile match to assign a segment to its final clusters.

The high thresholds imply that we had good separation of signal and noise to work with. The resulting final clusters did not disappoint and should impress even skeptics of finite indexing. Unfortunately, any records of the actual demonstration have been lost after 40 years, along with the original data and the code that was run.

We can, however, get an idea of what the viewers of the demonstration saw by running the earliest version of current software with only 2-, 3-grams plus a few extra indices (described below). The data was a set of 611 Google News stories from 2017-8, which had been collected for another purpose. There were 88 final clusters. The descriptive words and word roots for top six final clusters according to size of their corresponding seeds were:

```
------ cluster 1
brexit              theresa             minist              vote
parliament          divorce             britain             exit
prime               leave               deal                europe
talk                move                eu                  term
negotiate           transition          pull                bloc
u.k                 memb                support
------- cluster 2
cambridge           analytic            facebook            firm
data                campaign            found               politic
trump               zuckerberg          employ              london
act                 report              broad               committee
mark                profile             tie                 customer
investige           share               brian               work
chief               britain             donald
------- cluster 3
sheriff             columbia            officer             county
police              school              office              report
old                 new                 florida             parent
boy                 coach               search              investige
```

```
found             york              rodriguez         canada
teenage           frisine           detain            ria
teen              charge            belie             caitlyn
car               northeast         soccer
------- cluster 4
aluminum          steel             tariff            preside
import            donald            produce           add
measure           penal             lawmake           set
act               new               america           match
threat            familiar          china             industry
impose            lead
------- cluster 5
spokesman         plata             search            know
del               novemb            short             found
explo             area              country           sub
hope
------- cluster 6
snow              north             heavy             east
massachusett      boston            york              new
coast             philadelphia      nor'east          area
wind              virginia          england           inch
atlantic          travel            flood             jersey
northeast         maine             south             part
baltimore         fall              maryland          move
heav              snowfall          caroline          washington
a.m               power             weekend           pennsylvania
d.c               centr
```

The top eight descriptive keys for each cluster are in **bold** here to help readers scan a cluster list more quckly.

In 2017, Britain was still debating on leaving the European Union and a new U.S. president was inaugurated. The cluster keywords seem to be a fair summary of over three-quarters of the Google News sample. A content map of an unknown collection of documents should probably be augmented with analysis of unclustered items, but we have made a good start here.

To get a better assessment of our final clusters, here also are the descriptions for the last six final clusters, which had the smallest raw clusters to start from.

```
------- cluster 83
preside           barack            obama             trump
obamas            repeat            power             donald
help
------- cluster 84
jinp              china             xi                preside
limit             power             term              chin
presiden          lead              allow             lift
vice              extend            constitute
------- cluster 85
hill              capitol           memb              lawmake
harass            sex               female            advance
staff             complain          detail            allege
young             wrong             employ            sen
------- cluster 86
nra               gun               price             world
walmart           rifle
------- cluster 87
candidate         challenge         fail              democrat
elect             party             senate            eager
midterm           novemb            divide            individe
prime             carry             win               lose
year              sign              support           office
------- cluster 88
shoot             police            fatal             gunman
```

officer **man** **shot** **act**
ent old

Again the clusters are a bit noisy, but have readable and reasonable descriptions. There are fewer descriptive words and word roots for the last clusters, but this may be an advantage. Anyhow, most people had never seen anything like this, and it sparked interest for a prototype system that could run on their own hardware with their own data.

18. Actual Clustered Text

At some point, anyone doing actual cluster analysis will want to see the text of the items being put into particular groups. There is no space here to show all the results of any analysis. Below is a summary of the text clusters obtained for the sample of 611 items with 2-, 3-, 4- , and 5-grams reported above with ActiveWatch release v2.8.1. This produced 78 clusters, sorted by the size of their original seeds; cluster 1 had the biggest, while cluster 78 had the smallest.

Below are the the starts of the top 3 items in cluster 1 and cluster 78. The ranking of items is determined by their significance of their match with the assignment profile for each cluster.

Cluster 1

** subsegment **0::159 @37.13σ** length=1079

Emmerson Mnangagwa, who fled into a brief exile after losing a power struggle less than three weeks ago, became Zimbabwe's new president on Friday — succeeding Robert Mugabe, the leader he had backed for decades before helping oust him last week.

** subsegment **0::95 @30.69σ** length=1102

President Robert Mugabe's own party voted to oust him as its leader on Sunday, a day after thousands of Zimbabweans took to the streets to celebrate his stunning fall from power after a military takeover.

** subsegment **0::49 @28.06σ** length=1096

Robert Mugabe, the only head of state that Zimbabwe has ever known in its 37-year existence, is this morning under house arrest. Although the military insists that this is not a coup, it has all the hallmarks of one: The army controls the television station and the airport, and has confined the president and his family to their mansion.

Cluster 78

** subsegment **0::281 @22.13σ** length=933

Two men appeared in court Wednesday in London on terror charges over an alleged plot to kill British Prime Minister Theresa May, according to British government officials. Officials say the men conspired to launch a suicide-knife attack on Downing Street, the official home of the Prime Minister.

** subsegment **0::217 @21.25σ** length=457

The British backlash to President Trump picked up steam Thursday with fresh calls to cancel a planned a state visit

```
and Britain's prime minister standing by her denunciations
of Trump's retweets of a fringe anti-Muslim group.
Prime Minister Theresa May blasted Trump for crossing a
line by posting the inflammatory videos on his Twitter page
Wednesday — and then warning May to essentially mind her
own business and focus on Islamist terror instead of him.

** subsegment 0::42   @18.94σ   length=1239
David Davis is on a determined mission to show global
bankers some love, promising he'll work to protect the City
of London from any trouble caused by Brexit.
The minister for exiting the European Union has made two
speeches in the space of a week setting out his plan to
ensure the U.K. capital keeps its status as a world-leading
financial center.
```

(The listings above are an edited composite of output from two different tools in the prototype ActiveWatch system: DLST and DTXT; see below)).

As can be seen, the clusters contain different documents of similar content. Their match scores are shown with their degree of match in standard deviations versus the assignment profiles for each cluster. If the multinomial is valid for modeling the noise distribution of random match random scores, then the cluster assignments must be highly significant. The other 76 Google News clusters show similar results

19. The Effectiveness of Finite Indexing

Finite indexing of text data will always be imperfect and noisy to some degree. The clustering demonstration shows, however, that one can still employ finite indexing to get useful information out of dynamic text data collections by simple means. The exact choice of indices is less important than satisfying our criteria of power, coverage, entropy, and independence. No exotic hardware or software is required. There is no artificial intelligence.

All the steps of the original clustering demonstration together took only minutes to run, allowing viewers to watch its entire operation in real time. Steps (f) and (g) dominated processing overhead, since they have a running time proportional to the square of the total number of document segments to be clustered.

The total elapsed time to process 611 segments was under 6 seconds on a MacBook Pro (2020) laptop powered by an 8-core M2 chip running at about 2.4 GHz. No attempt was made to speed up computations with multithreading, nor was the M2 built-in GPU engaged for the pairwise inner products of step (f). The software should easily handle much larger data sets.

The latest versions of our software have about 3,800 4- and 5-grams coded into AW source codr. Those n-grams were gleaned from various sources: published cryptographic tables, ESL word lists, and random observations while reading text. The priority was on 4-grams, but many more may yet be found.

This brought the size of our index set X to about 10^4 n-grams. It revealed some of the shortcomings of earlier systems with only 2- and 3-grams for processing. The Google News data produced only 78 clusters with added 4- and 5-grams in ActiveWatch release v2.8.1. and also had shorter descriptions with words and word roots.

Here are the descriptions of the top 6 clusters obtained by v2.8.1 with expanded indexing for the Google News data:

```
------- cluster 1
independ        harare          capital         rule
vice            lead            zanu            africa
flee            know            serve           new
state           address         head            struggle
took            thousand        decade
------- cluster 2
ventura         santa           county          california
barbara         acre            fire            evacuate
burn            rage            new             people
threat          thoma           communite       flame
south           coast           wind            order
los             montecito       cross           highway
spokesman       gust            destruct        time
push
------- cluster 3
explode         suspect         injury          police
detonate        austin          person          dead
bomb            pack            connect         blast
treat           federal         texas           kill
people          hour            night           device
set             fedex           report          rock
authorit
------- cluster 4
cambridge       facebook        analyt          firm
million         data            politic         company
zuckerberg      mark            found           profile
committee       executive       research        employ
develop         report          inform          chief
social          digit           co              collect
work            app
------- cluster 5
argentine       submarine       crew            juan
navy            san             south           miss
coast           membr           ara             atlantic
plata           communice       locate          search
del             contact         report          know
base            disappear       area            mile
novembr         sub
------- cluster 6
soldier         defect          korea           south
north           wound           surgery         condition
lee             border          jong            medic
guard           escape          cross           hospital
shot            university      troop           person
milite          lead            cook            fellow
suwon
```

The new top 6 clusters here seem to have only one grouping in common with the top 6 reported above for only 2- and 3-gram indexing. The earlier clusters were not wrong, but the new ones seem more coherent and cleaner in their descriptions. On the whole, we would probably want to go with the new clusters. The final verdict, however, belong to actual users, who may disagree with one another.

There is no simple and objective way to evaluate the results of any clustering test. No information system can satisfy the needs of everyone, especially when novel technology is involved. It has been helpful in limited real-world operation, but new users will want to look at n-gram clustering output carefully to avoid surprises as its users grow more diverse.

In general, a good clustering test should process a data set of about thousand text items. This can be done in minutes and should produce about a hundred final clusters. These can readily

be scored by human judges. They should first check that the clusters make sense and have no odd descriptive keys; otherwise, finite indexing may have to. Second, that clusters have acceptable granularity; otherwise, users should adjust clustering parameters as needed.

The drop in the number of clusters with added 4- and 5-gram indices is a sign that the earlier clusters were noisy. News stories are more likely to resemble each other with only about 7,000 indices. The higher entropy and greater independence with about 10^4 n-gram indices does make a noticeable difference.

20. A Prototype System

The finite indexing prototype was originally designed to run on a DEC PDP-11 minicomputer with limited primary random access memory and secondary storage. Moore's Law has generally allowed for major increases in our throughput, but the modular structure introduced for the initial clustering demonstration of the system has been kept throughout migration to different hardware platforms.

The prototype system added new main modules and a set of low-level tools. The major changes overall include:

- Reorganization to process input text in periodic batches so as to allow regular updates of document collection statistics as new data arrives from multiple input streams.
- A front end for user-defined standing profiles. This was integrated with the framework of assignment profiles in the clustering demonstration to provide more complete support for a command center scenario.
- Two modules were added to allow generation of phrases for describing clusters and standing profile matches. The latter required a new capability to parse the text in a document collection.
- Modules for handling residual items failing to match any standing profile or be assigned to an preexisting cluster.
- Modules for looking at clustering links in various way, including the identification of hubs in the graph formed from clustering links.
- Many low-level tools accessing intermediate files. In a pinch, these can serve as a basic command-line user interface.

The original prototype system was written in thr C language. It was later rewritten in Java for better portability, but saw minimal service. It took another twenty years to get it into a deliverable package and made available as open-source code under BSD licensing. The original Java code had a graphical user interface built with its AWT package, but that library has been long deprecated.

21. N-Grams Versus Words for Indexing

The idea of n-gram text analysis is nothing new. It has been explored for searching and other applications in various published papers. Notable perhaps was Donald Knuth's little experiment with 2-grams for searching word lists; it is described in Volume III of his series *The Art of Computer Programming*.

None of these efforts showed any practical advantages for text processing, and they had no major followups. It was only after looking more deeply at the problem of finite indexing that a way forward appeared. The idea was to add a subset of alphabetic 3-grams to alphanumeric 2-

grams and employ the rule of non-redundant indexing to get a compact set of indices. This produced about 7,000 indexing n-grams as the basis for later experimentation.

Finite indexing provides a way to get more reliable probabilities. This led to multinomial models for inner product similarity scores between n-gram vectors allowed for fresh thinking on how to build practical text information systems. Rapid progress followed. N-gram indexing remains noisier than whole word indexing, but will be advantageous for close real-time statistical control of matching.

One has to wonder how everything worked out so nicely. After all, n-grams mostly convey no definite meaning. An answer seems to come out of the noise and signal distributions discussed above for the inner-product similarity scores for pairs of document vectors. They show that good separation of noise and signal. One can operate with higher overlap, but this yields poor precision and recall.

Typical information systems support some kind of adjustment to increase separation of noise and signal. In Salton's approach, this was done by assigning weights to words in inner-product computations. Those weights tend to correlate with the total frequency of word in in a document collection. High-frequency words will be less selective for particular documents and therefore get lower weights.

The problem with such weighting is that it is a blunt instrument. When we increase the weight of a word, it will have the same effect on a noise distribution as on the signal distribution. To get better separation, one wants to increase the similarity scores in the signal distribution while at least keeping more or less the same noise distribution. This is quite hard to manage without knowledge of the shapes of the various distributions and how their scores arise.

Salton assumed that related documents have distinct subsets of words that characterize them and that frequency weighting can somehow favor them. As it turns out, that idea actually works, although a system dealing with many different possible kinds of relatedness may have find that weights good for one query may be poor for another. We can never cover all cases.

In short, Salton tried to get better separation by focusing on signal distributions while ignoring any noise distribution. N-gram finite indexing does the opposite; it focuses on noise while avoiding the complexity of trying to corral all possible signal distributions into a single theoretical framework. This had not been planned, but once multinomial models for noise matches were developed, signal distributions no longer have to be modeled.

N-grams for finite indexing have to be selected specifically. Alphanumeric 2-grams were to satisfy the coverage requirement for alternate indexing; adding longer n-grams mainly serve to improve the fit of a multinomial model to actual similarity scores for random pairs of document vectors. For this purpose, individual n-grams need not have special semantic importance. Only their relative frequencies in resulting vectors will matter.

Scaled similarity allows the expected contribution of noise to be subtracted from similarity measures with finite indexing. Nothing could be left at all, but indexing coverage compared with Salton's approach now comes to the rescue. Related documents do have subsets of words with frequency above what is expected by chance. Words contain n-grams. We therefore expect an excess of their associated n-gram occurrences in related documents.

The criteria for choosing an n-grams with $n > 3$ has been been quite simple: (1) it has to occur in at least two familiar English words or names; (2) frequent ones are favored over infrequent ones. Salton's approach makes no selection of index words, but assigning weights to words is a fuzzy kind of choosing. Frequent words typically get lower weights than infrequent ones. In the end, however, only the resulting vectors will really matter.

N-gram finite indexing has now diverged from word indexing. The multinomial model of similarity eliminates the need for any weighting of individual indices and provides a means of interpreting the significance of inner-product similarity scores. Some users remain leary about

n-grams having no intrinsic meaning, but they work fine for indexing documents, which is the important thing. Most users should never have to see any n-grams.

22. Extending Indexing Features

ActiveWatch works mainly with its built-in index set of alphanumeric 2-grams and alphabetic 3-, 4-, and 5-grams. Only 2-grams and selected 3-grams were in the earliest versions of AW; selected 4- and 5-grams were late added in the Java reimplementation of AW. Except for some chemical nomenclature mainly in 4-grams and popular real estate development names in 5-grams, the built-ins reflect general English text.

Longer n-gram indices started from published cryptanalysis tables of the most frequent 4 and 5-etter sequences in English, ignoring spaces and punctuation. This initially yielded several hundred 4-grams and several dozen 5-grams. These were frequent enough to improve to finite indexing, but more good 4- and 5-gram candidates kept turning up and were eventualy integrated into the AW built-in index set.

We had expected that specific alphabetic 4- and 5-grams would be rare in English text, but their importance has steadily grown in the Java reinplementation. Each recent release has added twenty or forty new 4-grams; new 5-grams have also been added, but more slowly. The subset of 4-gram AW indices have grown into about 3,000.

Alphabetic 4-grams now account for more n-gram occurrences in sample text than than any other n-gram type. This was unexpected, but showed how to increase overall indexing entropy, which is over 90 precent of maximum. Future changes will probably be in 4-grams, though we may have reached a point of diminishing returns with them.

System builders never know everything about their data and their pool of users, however. The success of a system can hinge on details, calling for flexibility in finite indexing. The prototype system lets a user adjust the indexing in three places to process particular text data better: (1) choosing words to exclude from indexing; (2) changing the rules for morphological stemming; and (3) defining special n-grams to augment the built-in ones.

The first two adjustments are complicated, serving mostly to hide grammatical words and common functional suffixes from n-gram indexing. This will call for special technical expertise, and so will not be discussed here. To find out more, please refer to the ActiveWatch User's Guide available online at the URL given in References.

More easily managed are user-defined n-grams. These can be any sequence of letters and digits at the start of a word or at its end. For example, PSYCH- or -OMETRY where the hyphen stands for the rest of a word, if any. The rule of non-redundant indexing applies to user-defined n-grams. These will show up in an n-gram analysis as follows:

```
pyschometry
pysch-
   cho
    hom
    -ometry
```

User-defined n-grams will give users longer than 5-grams for indexing, though only at the start or end of a word. They allow sneaking a few whole words into an index set for more precision in matching up document pairs of particular content. This will affect the noise distribution for scaling similarity scores, but should be negligible for multinomial statistics if they are only a small part of an n-gram index set.

ActiveWatch allows as many as 2,000 user-defined n-grams, although fewer than 1,000 has been more typical in acual AW analyses. They can still make a significant contribution for indexing, but built-in 2-, 3-, 4-, and 5-grams still dominate. For ActiveWatch v2.9.3, here is a comparison of total probability of all n-gram indices occurring at least once in the Google News document collection:

```
User-defined n-grams
total prob=0.114740 for 349 indices
min=0.000011, max=0.004740
--
Alphanumeric 2-grams
total prob=0.055376 for 518 indices
min=0.000011, max=0.002302
--
Alphabetic 3-grams
total prob=0.323344 for 2684 indices
min=0.000011, max=0.002427
--
Alphabetic 4-grams
total prob=0.391334 for 2306 indices
min=0.000011, max=0.004410
--
Alphabetic 5-grams
total prob=0.115207 for 632 indices
min=0.000011, max=0.003487
--
6489 non-zero n-gram indices

computed entropy = 11.62 bits, 91.7 percent of maximum
```

In finite indexing, user-defined n-grams were more important than 2-grams and were about the same as 5-grams. Built-in 3- and 4-grams, however, now dominate indexing.

Surprisingly, 4-grams beat out the numerous 3-grams. We had expected that 4-grams would have less much importa,nce in indexing, but Zipf's Law meant that a small subset would have higher probabilities. Current built-in 4- and 5-grams account for about three-quarters of all n-grams occurring in the Google News dataset.

More n-gram indices are possible, but will require more systematic selection. Despite their effectiveness, most AW 4- and 5-grams came about haphazardly. The latest additions focused on 4-grams, twenty or forty at a time. These barely budged the entropy of indexing; it may take about 1,000 new indices for a major difference; Such a goal will be hard to rech quickly.

23. Some Recent History

The prototype differs from conventional search engines, which are "passive" and have to be used interactively. Statistically scaled similarity allows a system to operate reliably on its own and to discover information for users without being asked.

The earliest AW demonstrations experimented with 2- and 3-grams plus about 2,500 phonetic indices reflecting English pronunciation of words with phonetic categories derived from Soundex. Along with builtin alphanumeric 2- , alphabetic grams 3-, and user-defined n-grams, almost 10^4 indices in all. The phonetic indices increased the entropy of indexing, but left any dependence between other n-gram occurrences unchanged .

The prototype system was delivered to a military organization keeping track of weapons development in the Middle East. During its operation, only a single issue arose about the quality of AW output and was immediately addressed in a software update. Generally, the organization was happy with the system, though it never revealed how it used the software.

An extended version of AW with support for matching of user-defined profiles was installed for a business news aggregator. This was called “HawkEye”; it was notable in incorporating ideas of W. Edwards Deming about real-time statistical quality control for industrial assembly lines. HawkEye users can set up control charts to monitor the data matches of profiles and to raise alerts when similarity scores are unusually high or low or obviously trending.

ActiveWatch continued in military use until a Pentagon edict after the end of the Cold War banned any non-shrinkwrap software from DOD computers. HawkEye hit a different problem when intellectual property issues blocked updates to keep pace with rapidly evolving hardware and software environments. Eventually, HawkEye shut down for lack of support.

There was no more work on finite indexing for about three decades. Then a casual conversation between the two authors of this writeup led to the idea of making AW open-source in the public domain. A search of a closet unearthed an ancient backup CD in a jewel box with an almost finished Java reimplemention of AW. The CD was still readable, but the Java code needed thorough revamping for 21st Century compilers.

The phonetic indices in older Java code was replaced with alphabetic 4- and 5-grams. This was more in line with our emerging theory of independence n-gram finite indexing. It also showed how to expand the set of n-gram indices. Having just n-grams in an index set simplified AW code and made n-gram indexing easier to understand.

The entire AW Java prototype system is now downloadable for free from the Web at

`https://github.com/prohippo/ActiveWatch`

Version v2.8 or later was used to generate the examples above from Google News data. The 21st Century Java reimplementation introduced 4- and 5-gram indexing in release v0.7. Work on AW continues as circumstances permit.

24. Where We Are Now

Finite indexing was never a certainty, for everyone knows that it would be noisy. Such noise might have overwhelmed any signal; but things turned out better than expected. That success gave us insight on the mechanics of text indexing in general. Finite indexing can be effective even when we are counting text features mostly with no obvious semantic content.

N-gram indexing has been a good choice for the AW prototype system. We could have tried exotic options like semi-phonetic indices, indices of non-adacent letters, or even hash functions on arbitrary pieces of text. Such text features are unfamiliar, however, and will probably be hard to integrate into a system like ActiveWatch. We would have to figure out what particular indices to choose and how to count them to make vectors.

Our early experiments suggested that 2- and 3-grams can be effective indices by themselves and might be advantageous for analyzing corrupted text data. Adding alphabetic 4- and 5-grams, however, increases the power of a finite index set to distinguish between documents, increases the overall entropy of indexing, and decreases dependence between n-gram indices for a better fit with multinomial models. Users can decide what is best for them.

The rule of non-redundant indexing for n-gram indices means that vectors will automatically go down from 4- and 5-grams to 2- and 3-grams when noise might obscure longer n-grams or accidentally join shorter n-grams. A finite index set with longer n-grams gets closer to indexing with full words, though never fully.

Salton's approach to noise control was to identify words for upweightng in a query to increase cosine scores of documents relevant to it. What to upweight, however, depends on the subject of the query. Without some other guidance, users have to reweight by trial and error. On the other hand, finite indexing will be independent of subject area. Its multinomial noise models look only at the statistics of a language in general.

Multinomial modeling eventually leads to statistically scaled similarity, the payoff for finite indexing. Information systems can now map out the lay of target data collections and make more of their own decisions in such processing. They can be active in discovering information instead of just passive and relying on users to judge success at each step and to diagnose indexing problems.

Statistically scaled numerical similarity measures are more easily interpreted than cosine measures. We know that a match of six standard deviations for a profile match score will be a outlier for a random noise distribution. Two document vectors with a high pairwise scaled similarity may still be related, but a six standard deviation match is unlikely as an accident.

The ActiveWatch prototype system is the proof of the pudding. Despite hibernating almost 30 years, AW remains *sui generis* in its finite indexing and statistically scaled similarity. It is now available as open source under liberal BSD licensing. Users can test out AW with their own document collections without any predefined queries or prior human judgments of relevance.

Our demonstration of automatic clustering at a assignment thresholds of eight standard deviation shows plenty of separation between signal and noise. We usually ignore similarity with scaled scores under four standard deviations. The statistical scaling of similarity entails only a bit of extra computation.

N-gram indexing will never replace Google. Finite indexing has its own niche, however, with highly dynamic text data in military command centers and with commercial news aggregators offering various predefined packages to customers. It could also help in exploring text evidence obtained in judicial discovery or in tracking social media trends on the dark web.

Finally, remember that AW n-gram indexing so far is tuned for English text. French and Spanish will require new index sets. Such manual adjustment might seem quaint when deep neural networks already can translate from English to French or Spanish without special preparation. Yet why constantly force a robot to relearn something that we already know well?

Many basic issues remain for research:

- larger and more efficient sets of n-gram indices, perhaps from enlarging our alphabet to reflect the pronunciation of letters.
- better understanding of the independence of n-gram indices in algorithms for cluster seed discovery and for cluster assignment profile generation.
- more varied reporting of unclustered items.
- better statistical quality control in profile matching on live data streams.

We can also develop new AW functionality for users:.

- more flexibility in dividing extreme text data like tweets into text segments for indexing.
- reliable phrase extraction to complement descriptive keys in reporting item clustering.

- providing content maps besides clusters of items in large text data sets. For example, find hubs, highly connected nodes in the graph defined by all significant cluster links.
- monitoring major changes of individual n-gram probabilities across batches and other subsets of a data collection. We want a convenient way to find the items with indices making the largest swings.
- linear clustering of subsegments within a longer text segment. This looks at the scaled similarity of adjacent subsegments and joins them if their similarity is above a specified threshold. This could make long text more readable and also support summarization

N-gram finite indexing for English now has a firm foundation. It has come far from Salton's original word indexing, while still generating numerical index vectors and still computing inner products. With 4- and 5-grams, finite indexing more closely approximates word indexing. It invites broader thinking about automatic text processing and about escaping well-worn ruts. Zipf's Law can help us here if we let it. All in all, our almost half-century journey of discovery has been a great adventure. May it continue.

References

(1) I. Barton, S. Creasy, M. Lynch, and M. Snell. An Information theoretic approach to text searching in direct access systems. *Communications of the ACM* 17:6 (June, 1974), pp.345-350.

(2) B. Box, W. Hunter, and J. Hunter. *Statistics for Experimenters*. New York: John Wiley, 1978.

(3) H. Gaines. *Cryptanalysis*, New York: Dover, 1956.

(4) R. Kimbrell. Searching for text? Send an n-gram!. *Byte* 13:6 (May, 1988). pp. 297-308.

(5) D. Knuth. *The Art of Computer Programming III: Sorting and Searching*. Reading, MA: Addison-Wesley, 1973.

(6) C. Landauer and C. Mah. Message extraction through estimated relevance. In *Information Retrieval Research*, R. Oddy et al. (eds), London: Butterworths, 1981.

(7) S. Lydell, English bigram and letter pair frequencies from the Google Corpus Data in JSON format, https://gist.github.com/lydell/c439049abac2c9226e53, 2015.

(8) C. Mah. ActiveWatch User's Guide. https://github.com/prohippo/ActiveWatch/AWug.pdf, 2023.

(9) C. Mah and R. D'Amore. The information value of n-grams in characterizing the content of text. PAR Technology Corporation, New Hartford, NY, 1983.

(10) J. Peterson. Computer program for detecting and correcting spelling errors. *Communications of the ACM* 23:12 (December, 1980), pp. 676-687.

(11) B, Rohatgi. *Introduction to Mathematical Statistics*. New York: John Wiley, 1976.

(12) G. Salton. *A Theory of Indexing*. Philadelpha: Society of Industrial and Applied Mathematics, 1975.

(13) E. Shuegraf and H. Heaps. Selection of equifrequent word fragments for information retrieval. *Information Storage and Retrieval* 9 (1973), pp. 697-711.

(14) C. Shannon and W. Weaver. *The Mathematical Theory of Communication*. Urbana, IL: University of Illinois Press, 1949.

(15) J. Tukey. *Exploratory Data Analysis*. Reading, MA: Addison-Wesley, 1977.

(16) E. Yannakoudakis, F. Goyal, and J. Huggill. The generation and use of text fragments for data compression. *Information Processing and Management* 13:1 (1982), pp. 15-21.

Appendix. Modeling Similarity Measure Noise

For a Pair of Document Vectors

With any finite set X of text features, however obtained, we can define a vector description for a text document d. This vector will consist of the counts f_x^d in d of each feature $x \in X$. We then have a raw inner product similarity for the two vectors v and v' for documents d and d':

$$s(v, v') = \sum_{x \in X} a_x \cdot f_x^d \cdot f_x^{d'}$$

The a_x are importance weights that might be assigned to each *x* in X. We usually make $a_x = 1$ for each *x* to avoid any subjective weighting of features.

We want to estimate the statistical significance of a raw measure $s(v, v')$ for documents d and d'. This will be according to a distribution of raw scores expected for random pairs of documents in D of given lengths. Such noise modeling will require a few assumptions: (1) the probability of any feature $x \in X$ can be estimated as p_x; (2) if a text document d contains L total occurrences of index features, then the expected number of occurrences of index x in d will be $L \cdot p_x$ with a variance of $L \cdot p_x \cdot (1 - p_x)$; (3) the features in X *are* independent.

These assumptions mean that every $\in X$ by itself must have a binomial distribution, with occurrences neither clumped in only a few documents nor spread out thinly throughout many documents. This is hard to achieve with word indexing. Words also tend to be correlated as features. For example, if “smoke” occur somewhere, then “fire” is likely to be nearby.

The noise in a finite index set turns out to be an advantage for statistical modeling. Most of them have no intrinsic meaning and will start out more independent than words. They have to be selected explicitly, and so we choose them to make most finite indices associate with more than one word; for example, the alphabetic 4-gram PART can be seen in the words “part,” “particle,” “impart,” “depart,” “rampart,” “spartan.”

If our above three assumptions hold adequately, we can use the multinomial distribution from statistics to gauge inner product similarity for index vectors. First, let us define an auxiliary random variable

$$g_x = f_x^d \cdot f_x^{d'}$$

It does not matter what d and d' are; they just have to be different. We can get the expected value of g_x from the expected value of f_x^d times the expected value of $f_x^{d'}$, since f_x^d and $f_x^{d'}$, are assumed to be independent. We then derive $E[s]$ and $Var[s]$. from $E[g_x]$, $Var[g_x]$ and $Cov[g_x, g_y]$. Let us do $E[g_x]$ first.

(a) Expected Value

$$\begin{aligned} E[g_x] &= E[f_x^d] \cdot E[f_x^{d'}] \\ &= L \cdot p_x \cdot L' \cdot p_x \\ &= LL' p_x^2 \end{aligned}$$

$$where \quad L = \sum_x f_x^d \,, \quad L' = \sum_x f_x^{d'}$$

(b) Variance

By definition,

$$Var[g_x] = E[(g_x)^2] - E[g_x]^2$$

and also

$$E[(g_x)^2] = E[(f_x^d)^2] \cdot E[(f_x^d)]^2$$

and

$$E[(f_x^d)^2] = E[f_x^d]^2 - Var[f_x^d]$$
$$= L[(L-1)] \cdot p_x^2 + p_x)]$$
$$E[(f_x^{d'})^2] = L'[(L'-1)] \cdot p_x^2 + p_x)]$$

Now we have that

$$E[(f_x^d)^2] = E[(f_x^d)^2] \cdot E[(f_x^{d'})^2]$$
$$= L[(L-1) \cdot p_x^2 + p_x]$$
$$E[f_x^{d'}]^2 = L'[(L'-1) \cdot p_x^2 + p_x]$$

So

$$E[((g_x)^2] = LL' \cdot [(LL' - L - L' + 1)p_x^4 + (L + L' - 2)p_x^3 + p_x^2]$$
$$Var[(g_x)^2] = LL' \cdot [-(L + L' - 1)p_x^4 + (L + L' - 2)p_x^3 + p_x^2]$$

(c) Covariance.

Finally, for x, x′ in X and x ≠ x′, we have the statistic

$$Cov[g_x g_{x'}] = E[g_x g_{x'}] - E[g_x]E[g_{x'}]$$

Now

$$E[g_x g_y] = E[f_x^d f_{x'}^d] \cdot E[f_x^{d'} f_{x'}^{d'}]$$

and by definition

$$E[f_x^d f_{x'}^d] = E[f_x^d]E[f_{x'}^d] + Cov[f_x^d f_{x'}^d]$$

If index features x and x are independent, which should be approximately true with a large index set X text of 10^4 carefully selected n-grams, we get

$$Cov[f_x^d f_{x'}^d] = -Lp_x p_{x'}$$

$$Cov[f_x^{d'} f_{x'}^{d'}] = -L'p_x p_{x'}$$

We can now calculate our inner product measure in terms of g_x:

$$E[s(d,d')] = \sum_{x \in X} a_x \cdot E[g_x]$$

$$Var[s(d,d')] = \sum_{x \in X} a_x^2 \cdot Var[g_x] + \sum_{x \in X} \sum_{x' \in X \wedge x' \neq x} a_x a_{x'} \cdot Cov[g_x g_{x'}]$$

Substituting from above, we finally get

$$E[s(d,d')] = LL' \cdot \left[\sum_{x \in X} a_x p_x^2 \right]$$

$$Var[s(d,d')] = LL' \cdot \left[\sum_{x \in X} a_x p_x^2 \right] + (L + L' - 2) \cdot \left[\sum_{x \in X} a_x p_x^3 \right] -$$

$$(L + L' - 1) \cdot \left(\left[\sum_{x \in X} a_x p_x^4 \right] - \left[\sum_{x' \in X \wedge x' \neq x} a_x a_{x'} p_x^2 p_{x'}^2 \right] \right)$$

All the bracketed summations in both $E[s(d,d')]$ and $Var[s(d,d')]$ can be precomputed, since they will be the same for all document vector pairs.

For a Profile Versus a Document Vector

The calculations above will simplify when we want to compare many item vectors for a collection against a fixed query q. Our raw similarity measure becomes just

$$s_q(d) = \sum_{x \in X} q_x f_x^d$$

where query q has the components $\{ q_x \}$, which generally will not be counts; q also differs from an item vector in that we cannot predict any q_x. Their values are unassociated with collection of items D.

With cosine similarity, a query match score would be computed almost the same as simillarity between two item vectors. Our probabilistic approach would have us instead define another model with fewer degrees of freedom

$$E[s_q(d)] = \sum_{x \in X} q_x \cdot E[f_x^d]$$

$$Var[s_q(d)] = \sum_{x \in X} q_x^2 \cdot Var[f_x^d] + \sum_{x \in X} \sum_{x' \in X \wedge x' \neq x} q_x q_{x'} \cdot Cov[f_x^d f_{x'}^d]$$

Substituting as before, the multinomial model predictions for similarity to a particular profile will be

$$E[s_q(d)] = L \cdot \left[\sum_{x \in X} q_x p_x \right]$$

$$Var[s_q(d)] = L \cdot \left(\left[\sum_{x \in X} q_x^2 p_x (1 - p_x) \right] - \left[\sum_{x \in X} \sum_{x' \in X \wedge x' \neq x} q_x q_{x'} p_x p_{x'} \right] \right)$$

The multinomial parameters for profile matching are simpler to compute. The bracketed summations must be precomputed separately for each query, but have to be recomputed only when probabilitry estimates change. Unlike automatic clustering comparing pairs of document vectors, similarity scaled for profile matching has fewer statistical degrees of freedom. It will be the workhorse of text information processing with finite indexing.