

A Theory of Finite Indexing

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

1. Introduction

In 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 documents. This is hard for computational systems to access because language structure in text is complex, highly unpredictable, and often quite irregular. Different languages, or even dialects in the same language, can operate under different rules and will 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 ambiguous sentence particles.

Salton's insight was that most messy linguistic details can be ignored in text data applications needing to deal only with general content. To figure out what a text document is about, one can do well by just looking at what words occur in the document, regardless of how these words might be formed or how they might be 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 words in D and f_w^d is the number of times that a word $w \in W$ occurs in document d . Index set W is unbounded. Whenever new documents enter a collection, these often will contain words a system has never seen before.

Given a similar vector u for a second document e in D , we can compute a similarity score between them as inner product

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

This formula works only if documents d and e both have only finitely many words. The inner product here turns out still to be awkward to work with as a similarity measure. It is sensitive to the length of documents being compared and has no bound. Salton tried to avoid these problems by normalizing document vectors to a unity length in a Euclidean vector space.

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

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

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

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

where 1 is an exact match and 0 is none at all. The cosine measure is often used in document searches to get the best match in a collection D for a document d for a query q and also to cluster documents by similarity. Such algorithms typically benefit from setting some minimum similarity threshold on similarity to reduce computation. The cosine measure makes this harder because of inconsistent scaling for assessing the significance of similarities < 1 .

2. Finite Modeling

A standard engineering practice is to define a simplified model to expedite design computations for a major project. 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. Models can also help in problems like 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, which has had success in many information applications. It is important to remember, however, that any model is only one possible approximation of the real world. As systems evolve, they often reveal the shortcomings in their initial modeling, forcing re-examination of their assumptions. After almost a half century later, can we do better than Salton's approach?

One obvious shortcoming with Salton's vector representation is that the dimensionality of such vectors is effectively infinite. Since individual documents will always have a finite number of words, we can finesse this issue by computing with sparse vector encodings, but this is tricky in document collections with highly dynamic content. Its set of indexing features over time will always be growing and so must their vectors.

For example, deep neural nets were invented early in the 21st Century. These have led to tremendous progress in artificial intelligence through unsupervised machine learning from vector data, but the input layers of such networks will always have some fixed number M of nodes that all their input data has to match up with. With vectors growing uncontrollably in dimensionality, a network can quickly become obsolete.

As will be seen below, there are also other reasons for finite document vectors, if we can proceed carefully. To transform a vector counting occurrences of words $w \in W$ to a new vector counting occurrences of various $x \in X$, a finite index set, we have many options. The simplest is a linear transformation

$$T^M : W^\infty \rightarrow X^M$$

where T^M might be represented as a rectangular matrix with M rows of unbounded length. That can be hard to compute with, but we can choose a target index set X so that each row will have only a finite number of non-zero entries and so be represented compactly as a sparse vector. Any $d \in D$ must have a vector v^d of finitely many word counts and is transformable into a finite indexing vector by multiplying it as a column vector with our matrix for T^M .

$$T^M(v^d) = [T^M] \circ v^d$$

There are an infinite number of ways to define T^M . The challenge here is to find a definition with a finite index set X that can give us the performance we want.

3. An Alternative to Words for Indexing

Our discussion will now turn exclusively to text in English, as written with a Latin alphabet. In other languages like Chinese (ideographic), Korean (syllabic), or Japanese (ideographic plus two different syllabaries), constructing a finite set of indexing features will be quite different.

Let us first lay out some general guidelines for exploring finite indexing possibilities in English text. These will 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 a little basic information science and knowledge of English, we might 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 . This should be large enough to do almost as well as a sixth grader in distinguishing between different kinds of possible content.

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

$$v^d \neq 0 \rightarrow T^M(v^d) \neq 0^M$$

where 0 is the zero vector of Salton's W 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. When we have the mapping of infinite to finite vectors, this must introduce irreversible indexing noise.

entropy The frequency 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 often described as Zipf's Law, often expressed mathematically as

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

where F_x is the total frequency of feature $x \in X$ in a text data, and $\text{rank } x$ is the ordinal ranking of feature x in a data set (1 = first, 2 = second, and so forth). This law seems a bit strange, but does help 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 $p_x = p_y = 1/M$.

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

This would give us a maximum information entropy

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

The presence of the \log_2 means that entropy will be expressed as bits of information. Minimum entropy is when $p_x = 1$ for only a 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 still want our entropy as large as possible.

independence

This is a weakness of indexing text with words W . In English especially, words can be synonymous to a high degree; for example, RED, RUFOUS, SCARLET, CRIMSON, RUBY, CARNELIAN, or CARMINE. One might ignore such word synonymy for some text information applications, but in finite indexing, every $x \in X$ must pull its own weight; otherwise, the indexing power of X will be illusory.

Independence will also turn out to be quite useful theoretically. Ideally, we want any indexing pair $x, y \in X$ to have the property

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

where p_{xy} = the probability of x and y both occurring in a single document. Unfortunately 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 a sampling of text, they will be less reliable for rarer indices $x \in X$. So, our test for independence will have to be a looser inequality

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

for some small $\epsilon > 0$, which should be on the order of most p_{xy} . Zipf's Law tells us that, for most x , p_x will be quite small and that we can achieve more independence among them by making those p_x even smaller. This can be done just by increasing M . Zipf's Law will still trip us here, though, in that a few p_x will always be much higher than the rest.

We are close. Do we really need to satisfy our inequality above for all p_x ? Can we live with a few exceptions and work with incomplete independence? Will this good enough for our purposes? The only way to find out is to try out bigger X with actual data and see what happens.

We now have all the basics to launch finite text indexing. Most important is the ability to work with the probabilities $\{p_x\}$ because of only a finite number dimensions. The next step is to define a finite set X of text indexing features for English and to compute their probabilities.

4. The Probabilities of Index Features

Let us clarify what we mean by p_x . Finite indexing simplifies the estimation of probabilities. This will require only finite summing of the occurrences for each $x \in X$ of an indexing space over a large sample of text data. Our estimated probability p_x of feature $x \in X$ will simply be

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

The quality of estimates will of course depend on the collection D , but with a large enough D , they should be fairly close to those for English in general.

A finite set X with about 10^4 features would require a significant amount of computation for probabilities, but modern processors can handle such loads easily. When data collections are dynamic, we can even still afford to update probabilities often. Our current strategy is to use D itself as our sample data for estimates. We then process new text data in small batches and update probabilities after indexing each batch.

In Salton's system, a word index set W can grow without bounds, though at any time, it will remain finite. One still might estimate probabilities for $w \in W$ as above, but the arrays for keeping partial sums will grow without bounds. Also, most words in W will be rare, resulting in poor estimates of their probabilities. One typically would want at least 10 to 20 occurrences of an indexing features for best results. This is easier to accomplish with finite indexing.

Salton's system avoids working with any probabilities. Functions of the relative frequency of words like inverse document frequency do help to weight query terms; but they never really go anywhere else. Finite indexing, in contrast, will be built around probabilities. Even with estimation error, indexing probabilities will let us better assess the importance of individual text items and to get a handle on the significance of similarity scores for a pair of items.

5. Finite Indexing, Step by Step

The four requirements of Section 3 above mean that

$$X \not\subset W$$

since we would otherwise violate the consistency requirement for finite X . It is possible, however, to put a few whole words into a finite X . With text data in English, a better general indexing alternative overall is to count word fragments instead of full words. This will meet our consistency requirement, though possibly falling short for power, information, and independence. A word fragment approach will turn out to have important consequences,

Consider fragments of single letter (1-grams¹). 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 consistency requirement though, which is a start.

How about alphanumeric 2-grams? English has $36 \times 36 = 1,296$ 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.

¹ Google uses the term “n-gram” to refer to sequences of whole words. Our usage of the term is from cryptanalysis, which long predates Google.

```

resource
re
  es
    so
      ou
        ur
          rc
            ce

```

This 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 $36 \times 36 \times 36 = 46,656$ alphanumeric 3-grams. This seems out of control, since most will be nonsense like JRJ, HHQ, EEE, 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 consistency requirement, but add on selected alphabetic 3-grams to increase the power of our index set?

The idea is to count a 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

```

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 n -gram, we do not count any shorter n -grams falling entirely within that segment

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

In general, the addition of a single frequent $(n+1)$ -gram to an index set X consisting of k -grams ($k = 1$ to n) will be advantageous. The gain here will be cumulative, which suggests how we might get ultimately to 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 get $216 \times 26 = 5,616$ 3-grams, which will give us a little under 7,000 to index with, which should be good enough for initial experiments.

6. Adding a Key Theoretical Detail

At some point, we have to think about building practical software achieving important information goals. First, we must to run experiments to test our ideas objectively and rigorously. The results of such experimentation will be presented in subsequent sections, but we must begin by providing a proper 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, for it will be critical to deeper understanding of all text indexing. This will have no counterpart 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 using a multinomial model is that each $x \in X$ will have a frequency of occurrence across documents that have a binomial distribution, and p_x must be independent of p_y when $x \neq y$. This is hard to meet directly; for example, the occurrences of the most interesting words tend to clump in fewer documents than a binomial assumption would predict. The problem is less with n -gram indexing in that n -gram can occur across many different words.

With a bit of manipulation of random variables, a multinomial distribution lets us model the distribution of raw inner products between finite vectors. Its most important statistics will be

$$\begin{aligned}
 E[IP(d', e')] &= L_{d'} L_{e'} \cdot \sum_{x \in X} a_x p_x^2 \\
 Var[IP(d', e')] &= L_{d'} L_{e'} \cdot \left[\sum_{x \in X} a_x p_x^2 + (L_{d'} + L_{e'} - 2) \cdot \sum_{x \in X} a_x p_x^3 - \right. \\
 &\quad \left. (L_{d'} + L_{e'} - 1) \cdot \left[\sum_{x \in X} a_x p_x^4 - \sum_{x \in X} \sum_{y \in X \wedge y \neq x} a_x a_y p_x^2 p_y^2 \right] \right]
 \end{aligned}$$

where

$$d' = T^M(d)$$

$$e' = T^M(e)$$

$$L_{d'} = \sum_{x \in X} f_x^{d'}$$

$$L_{e'} = \sum_{x \in X} f_x^{e'}$$

$$a_x = 1 \text{ (by default)}$$

The full derivation of the expected value and variance of the raw inner product can be found in the Appendix. It assumes that the probabilities $\{p_x\}$ are independent. As noted above, this condition 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 they are independent enough.

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

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

This similarity score will be in units of standard deviations, easier to interpret than Salton's hyperspace cosines. We do not even have to approximate the multinomial noise distribution as Gaussian to interpret the significance of similarity scores. Current separation of signal versus noise, we let us go conservatively with the Chebyshev inequality to estimate p-values. In any event, we no longer have to normalize document vectors to a Euclidean length of 1.

7. Evaluation Framework

We now have all the ingredients necessary for some rigorous testing of finite indexing that counts 2- and 3-grams as features in text. 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, which will require extensive writing of actual software.

Salton evaluated the performance of document search with word indexing by running 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 execute 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 $r / (\text{total number of documents in } D \text{ relevant to the queries})$ 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 provides a more extensive test of a system. Such testing can be quickly done 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 vectors.

Here is how we now can do testing: Get 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 $v_A^{d'}$, where $L_{v_A^{d'}} = L$.
- For $d' \in D'$, obtain a different text segment to get an index vector $v_B^{d'}$ with $L_{v_B^{d'}} = L$.
- 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 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 evaluation procedure can be carried out objectively in only a week or two from scratch with any new data set. No human judges are needed.

8. 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 should already be familiar to most people working in text data indexing in general, but should be noted here because they will greatly affect the statistics in n-gram finite indexing.

The first tool relates to grammatical function words in Indo-European languages, including English. These help to structure 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 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 also 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 one needs 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.

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

9. Objective 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. With hardware extremely limited by small hard drives and random-access memory, information engineers had to think small.

The 2- and 3-gram subset in X was used in testing is described in the ActiveWatch User Guide. The 3-grams are somewhat inefficient, including many low-value 3-grams like THZ, ANB, and EEJ. We wanted, however, to come up with a reasonable X quickly. Our goal was only to show that n-grams could work; we were not reinventing the wheel. An X of about 7,000 2- and 3-grams is suboptimal, but would suffice for initial exploratory testing.

To recapitulate, there were only 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:

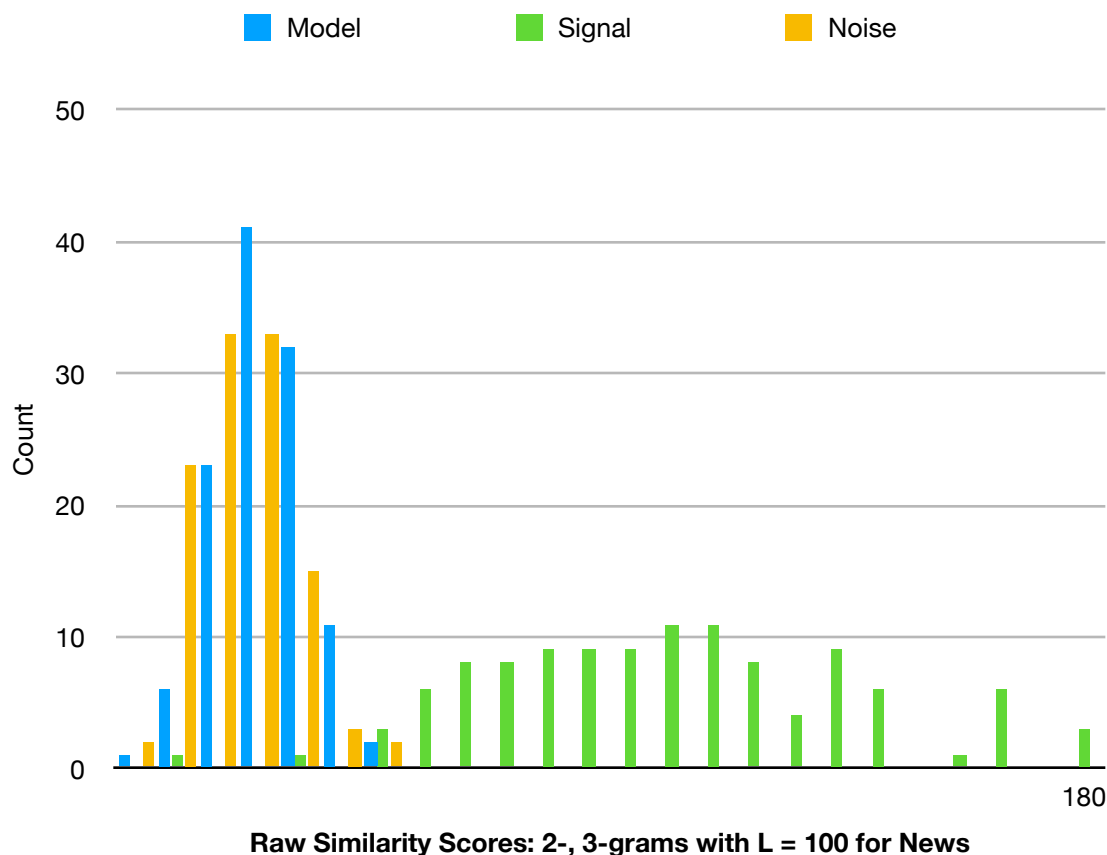
- Newswire stories collected for a previous project by a prospective sponsor (= News),
- Essays from the Norton Anthology, retyped (= Literary).
- A file of abstracts of technical papers in solid-state physics and electronic circuit design; these had been found in boxes of Hollerith punch cards (= Science).

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 any set of 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 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 similar 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 for us to start putting together a demonstration application based only on those n-grams.

10. 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. 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 effectively fit into 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 is the square-root 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 valuation 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 a 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	Comments
	Model σ	
News	7.0	the best
Literary	3.0	should be better with bigger L or M
Science	4.5	

11. A Demonstration Application

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 software. We must be 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.

$$q = \langle q_x \rangle_{x \in X}$$

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. If we do the same mathematics as for computing expected value and variance for pairwise similarity, we get

$$E[Raw(d, q)] = L_d \cdot \left[\sum_{x \in X} q_x p_x \right]$$

$$Var[Raw(d, q)] = L_d \cdot \left[\sum_{x \in X} q_x^2 p_x (1 - p_x) - \sum_{x, y \in X \wedge x \neq y} q_x q_y p_x p_y \right]$$

These noise distribution parameters must be recomputed for each new q ; for scaling document for pairwise similarity, we must recompute them only when 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 detecting them. The system always operates under the control of its users, but can run productively for extended periods of time without any intervention.

11. A Scenario 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 incorporate such a capability into a realistic system implementation and how it might extend what we have already in conventional search engines. Many text information users would seem to be already well-served by software now widely available on any desktop.

There are, however, some kinds of processing with more demanding requirements. Consider a command center gathering streams of data from dozens of different sources arriving day and night at unpredictable times. The incoming information is often highly perishable and might have dire consequences if not identified quickly. So, teams of personnel keep watch in shifts, which could be wasteful when data streams are quiet.

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 paper 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 how subjects are trending.

Such an operation is chaotic. No one person may see all related incoming information. Something can easily slip through, especially when arriving from multiple sources on different shifts to be pieced together. In modern command centers, everything is digital and stored on networked hard drives, but its basic problems remain. Technology can ease the burden of watch personnel, but will also increase the total amount of data to sift through.

12. 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 containing certain words, there is no reason to fragment those words and look for documents containing those fragments instead. This will only add

noise into search results. Places like command centers, however, need close 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 of all top results. This is acceptable where a searched collection is largely static. It gets more complicated when live data streams containing 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 coding and debugging skills. System employing numerical similarity scores for a match 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 should be easier to control. For example, matches 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 always show up in immediate search results; matches above 8σ will sound an alert. Notifications might also be set for when unreviewed matched 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 the type of clusters wanted; but around 8σ seems to work out well. A normalized, but unscaled, cosine measure will be much harder to adjust in these applications.

13. Demonstration Architecture

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

For this purpose, an application was built only to do 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 will 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 the 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 in the cluster that contain highly weighted n-grams in its profile.

This is much extra computation. It tests both of our two ways of scaling similarity scores in steps (e) and (g). Users should be able to see any problems in indexing of content here.

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.

14. 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 were not a disappointment and would 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 quickly.

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 in users for a prototype system that could run on their own hardware with their own data.

15. 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 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

16. 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. 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 differ from the original ones. It had about 3,500 4- and 5-grams from various sources, including personal linguistic competence. This brings 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 have only one cluster 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 to be more coherent and cleaner in their descriptions. On the whole, we would probably want to go with the new clusters.

The drop in the number of clusters with 4- and 5-gram indices added 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 improvement.

17. 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 processing 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. This 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 existing clustering
- 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 the C language. Later it was 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.

18. How N-Grams Differ From Words for Indexing

When the n-gram finite indexing project began, it was unclear if anything would come of it. The idea of using n-grams in searching and other applications had already been explored in various published papers. Notable perhaps was Donald Knuth's experiment with 2-grams for searching as described in Volume III of his series *The Art of Computer Programming*.

None of these efforts showed any practical advantages for text processing, and there was no major followup. 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 reliable probabilities. This led to multinomial models for inner product similarity scores between n-gram vectors and statistically scaled pairwise document similarity measures. That opened the way for fresh thinking on how to build practical text information systems. Rapid progress followed. N-gram indexing remains noisier than whole word indexing, but is advantageous for close real-time statistical control of matching.

One has to ask 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. The idea is that an information system should provide good separation of noise and signal. One can operate with high 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 every word in inner-product computations. Actual weights here tend to reflect the total frequency of word 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 approach 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 know 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 all have to be selected specifically. All alphanumeric 2-grams were included to satisfy the consistency requirement for alternate indexing, but the addition of 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, it matters not whether individual n-grams have special importance. Their relative frequencies in resulting vectors are what matter.

Scaled similarity allows the expected contribution of noise to be subtracted from similarity measures with finite indexing. There could be nothing left at all, but Salton's work now comes to the rescue. It seems that related documents do have subsets of words with frequency above

what is expected by chance. Words contain n-grams. We therefore should also expect an excess of their associated n-gram occurrences in related documents.

The criteria for choosing an n-grams with $n > 3$ has 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 system 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 really matter.

N-gram finite indexing became quite different from word indexing in the end. The multinomial model of similarity is essential. It 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.

19. User Adjustment of Indexing

System builders never know everything about their target pool of users. Yet the success of a system can hinge on details, and so some flexibility is required. In n-gram finite indexing. The prototype system lets a user adjust the indexing in three places to get better processing with particular text data: (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 somewhat complicated, serving mostly to hide grammatical words and common functional suffixes from n-gram indexing. Changing them will call for some technical expertise, and so will not be discussed here. For more details, 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. When these are in a indexing set X , the rule of non-redundant indexing still applies. They will show up in an n-gram analysis as follows:

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

User-defined n-grams will let users get longer n-grams for indexing, though only at the start or end of a word. They also are a way to sneak 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 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 is more typical. They can make a significant contribution for indexing, but 2-, 3-, 4-, and 5-grams still dominate. For ActiveWatch v2.8.2, 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=0.113775 for 349 indices
min=0.000011, max=0.004700
--
Alphanumeric 2-grams
total prob=0.055125 for 518 indices
min=0.000011, max=0.002282
--
Alphabetic 3-grams
total prob=0.332964 for 2708 indices
min=0.000011, max=0.002407
--
Alphabetic 4-grams
total prob=0.390597 for 2231 indices
min=0.000011, max=0.004372
--
Alphabetic 5-grams
total prob=0.107538 for 562 indices
min=0.000011, max=0.003457
--
6368 non-zero n-gram indices

```

User-defined n-grams had more impact on indexing than either 2-grams or 5-grams. The bulk of indexing, however, is still with 3- and 4-grams. Surprisingly, 4-grams beat out more numerous 3-grams, which justifies putting 4-grams into the AW built-in index set. Currently selected 4- and 5-grams account for more than half of all n-gram occurrences in a text dataset.

Further expansion of finite indexing is likely, but will require a more systematic approach with more text data. The last few additions of 4- and 5-grams have barely budged the total entropy of our n-gram index set. It takes about 1,000 more built-in n-grams to make a big difference. Even so, this will still be in the order of magnitude of our initial arbitrary 10^4 goal.

One can add user-defined n-grams whenever needed. These will be loaded at run time from a text input file (see the AW User's Guide). The dominant indices for our multinomial model, however, will probably remain 3- and 4-grams. Changes for finite indexing in subsequent system development will probably be mainly with 4- and 5-grams.

20. Recent History

The prototype finite indexing system was called ActiveWatch (AW), reflecting its conceptual roots in command centers. The "Active" adjective is to contrast the prototype's differences from conventional search engines, which are "Passive" and have to be used interactively. Statistically scaled similarity allows a system to operate with more autonomy and to discover information on its own.

The earlier AW demonstration application experimented with about 2,500 phonetic indices, which reflect the English pronunciation of words according to phonetic categories derived from Soundex. Along with 2-, 3-, and user-defined n-grams, this gives us almost 10^4 indices in all, but increased only the entropy of indexing. The inherent dependence between 2- and 3-gram occurrences was unchanged by phonetic indices.

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 exactly how it used AW.

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 on industrial assembly lines.

HawkEye users can set up control charts to monitor the data matches of profiles and to raise an alert when similarity scores are unusually high or low.

ActiveWatch continued in military use until a Pentagon edict after the end of the Cold War banned any non-shrinkwrap software from their 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 two decades. Then a casual conversation between the two authors of this writeup led to the idea of making AW open-source and in the public domain. A search of a closet unearthed an ancient backup CD-ROM in a jewel box with an almost finished Java reimplemention of AW. The CD-ROM was still readable, but the old Java code had to be thoroughly revamped for 21st Century Java compilers.

The phonetic indices in older code was replaced with 4- and 5-grams. This was more in line with our emerging theory of n-gram finite indexing. It also pointed the way to expanding the set of n-gram indices beyond 10^4 . Having just n-grams in an index set also simplifies the code and makes it easier to understand.

The entire AW prototype system is now available in Java for free download from the Web at

<https://github.com/prohippo/ActiveWatch>

Version v2.8 was used to generate the results above from Google News data. The Java reimplemention began when 4- and 5-gram indexing was introduced in release v0.7. The development of AW continues as time permits.

21. Conclusion

The exploration of finite indexing was never certain. In particular, we knew that n-gram indexing was inherently noisy. Such noise could have overwhelmed any signal; but everything turned out better than anyone had dared to expect. It led to a different way of viewing the mechanics of indexing in general. Finite indexing works, if we supply enough good text features to count.

Finite indexing is distinct from n-gram indexing. Our AW prototype system has shown, however, that n-grams are a good choice for finite indexing. Other options are possible; for example, phonetic indices, indices of non-adjacent letters, or even hash functions on arbitrary pieces of text; but they are more unfamiliar than n-grams and will take extra work to integrate into a multinomial model of noise distributions.

One can get respectable performance with only 2- and 3-grams, which might be useful for analyzing corrupted text data. The addition of 4- and 5-grams, however, increases the power of a finite index set to distinguish between typical 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 with n-gram indexing means that 4- and 5-grams will automatically bow out to 2- and 3-grams when high text noise prevents the longer n-grams from being found. If we know this beforehand, however, we can get somewhat more consistency in indexing by sticking to just 2- and 3-grams in the first place.

Noise has always been unavoidable for information systems, but often we can control it. Here, n-gram finite indexing departs from Salton's approach of weighting whole-word indices to raise the similarity measure for a pair of related documents, but what words to upweight will depend on the subject area. Finite indexing with n-grams is independent of subject; noise models for raw similarity need only look only at the statistics of a language in general.

Multinomial modeling eventually would lead to statistically scaled similarity, which is the payoff for finite indexing. Information systems can now make more of their own decisions in the

analysis of text data. They can be active in discovering information instead of just being passive and relying on users to judge success at each step.

People can also more easily interpret a statistically scaled numerical similarity measure. We know that six standard deviations for a profile match score should make it a far outlier for any random noise distribution. This is no guarantee that two matching document vectors with a high pairwise scaled similarity will actually be related, but odds are good that a match at six standard deviations is probably no accident.

The existence of the ActiveWatch prototype system is the proof of the pudding. Despite being in hibernation for over 20 years, it remains *sui generis* in its exploitation of finite indexing and statistically scaled similarity. Its availability as open source under liberal BSD licensing means that anyone can test it with their own document collections.

Our demonstration of automatic clustering at minimum thresholds of eight standard deviations shows plenty of separation between signal and noise. No human is required to evaluate the top matches in every ranking by similarity to make sure that the best in every case will actually be good. The statistical approach entails only a small amount of extra computation.

In a sense, our two multinomial models for document similarity measures turn indexing noise into a friend. Salton's concept of text indexing ignores noise, which is fine when an information user can closely observe a system and direct its operation. Such labor-intensive supervision is hard to sustain in real time in an age of streaming online information.

N-gram indexing will never replace Google or even turn researchers and developers away from Salton's hyperspheres in infinite-dimensional vector spaces. Finite indexing does have a niche, however, in the command center and in the aggregation of news for customers with special interests. Its robustness also may be helpful in exploring the content of text evidence obtained in pre-trial judicial discovery or in tracking social media trends on the dark web.

Finally, we should remember that n-gram finite indexing so far is tuned for English text. Even with French and Spanish, which have greatly influenced English, one must construct new n-gram index sets. Such AW specialization might be seen as unnecessary when deep artificial neural networks already can translate from English to French or Spanish without special preparation. Yet why force a robot to relearn something that we already know well?

Further work remains in n-gram finite indexing. Some immediate directions have already been touched on above, including:

- larger and more efficient sets of n-gram indices.
- better understanding of the independence of probabilities for overlapping n-grams.
- exploration of different algorithms for raw clustering and for generating n-gram profiles.
- providing more information in content mapping of large text data sets.
- defining more precise criteria for statistical quality control in profile matching on live data.
- setting guidelines for applying n-gram indexing to extreme text data like tweets.

The success of n-gram indexing so far for English was surprising. It invites us to think more broadly about the problems of automatic text processing and to get out of well-worn ruts. Zipf's Law can be a helper in the right context. All in all, it has been a great adventure.

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*. Philadelphia: Society of Industrial and Applied Mathematics, 1975.
- (13) E. Shuegraf and H. Heaps. Selection of equiprevalent 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

Given any finite set X of text features to index with, regardless of how these were obtained, we can define a vector representation for any text document d with the counts f_x^d of each feature $x \in X$. A raw inner product similarity measure for documents d and e can then be defined:

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

The a_x are importance weights that someone might assign to each index feature x in X . We usually make $a_x = 1$ to avoid any subjective weighting of features.

We still need to scale the measure $s(d, e)$ so that we can interpret similarity scores between pairs of text documents of differing lengths. Instead of converting their vectors to unit length as done for the cosine measure, we can approach this problem probabilistically with some simplifying assumptions: (1) the probability of any index feature $x \in X$ can be estimated; (2) given any text document d , with L_d occurrences of index features, x on the average will be expected $L_d \cdot p_x$ times with a variance of $L_d \cdot p_x \cdot (1 - p_x)$.

These two assumptions basically mean that the features of index set X are independent of each other. Strictly speaking, this is untrue. For example, words in text in particular do not occur independently; if you see “smoke” in some segment of text, then it becomes more likely that you will also see “fire.” Overlapping word fragments are also not entirely independent as features, but since almost all of them have no meaning by themselves, they will start out more independent than whole words.

If our assumptions hold approximately, we can employ the multinomial distribution in statistics to get a handle on our inner product similarity measure. First, let us define an auxiliary random variable

$$g_x = f_x^d \cdot f_x^e$$

It does not matter what d and e are; they just have to be different. We can compute the expected value of g_x from the expected value of f_x^d times the expected value of f_x^e , since f_x^d and f_x^e are assumed to be independent. We shall then derive $E[S]$ and $Var[S]$ from $E[g_x]$, $Var[g_x]$ and $Cov[g_x, g_y]$. We shall compute $E[g_x]$ first.

(a) Expected Value

$$\begin{aligned} E[g_x] &= E[f_x^d] \cdot E[f_x^e] \\ &= L_d \cdot p_x \cdot L_e \cdot p_x \\ &= L_d L_e p_x^2 \end{aligned}$$

(b) Variance

By definition,

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

To substitute above on the right, we need

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

where

$$\begin{aligned} E[(f_x^d)^2] &= E[f_x^d]^2 - Var[f_x^d] \\ &= L_d[(L_d - 1)p_x^2 + p_x] \end{aligned}$$

$$E[(f_x^e)^2] = L_e[(L_e - 1)p_x^2 + p_x]$$

Now we have that

$$\begin{aligned} E[(g_x)^2] &= E[(f_x^d)^2] \cdot E[(f_x^e)^2] \\ &= L_d[(L_e - 1)p_x^2 + p_x] \end{aligned}$$

$$E[f_x^e]^2 = L_e[(L_e - 1)p_x^2 + p_x]$$

So

$$\begin{aligned} E[(g_x)^2] &= L_d L_e \cdot [(L_d L_e - L_d - L_e + 1)p_x^4 + (L_d + L_e - 2)p_x^3 + p_x^2] \\ Var[(g_x)^2] &= L_d L_e \cdot [-(L_d + L_e - 1)p_x^4 + (L_d + L_e - 2)p_x^3 + p_x^2] \end{aligned}$$

(c) Covariance.

Finally, we have the statistic

$$Cov[g_x g_y] = E[g_x g_y] - E[g_x]E[g_y]$$

Now

$$E[g_x g_y] = E[f_x^d f_y^d] \cdot E[f_x^e f_y^e]$$

and by definition

$$E[f_x^d f_y^d] = E[f_x^d]E[f_y^d] + Cov[f_x^d f_y^d]$$

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

$$Cov[f_x^d f_y^d] = -L_d p_x p_y$$

$$Cov[f_x^e f_y^e] = -L_e p_x p_y$$

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

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

$$Var[s(d, e)] = \sum_{x \in X} a_x^2 \cdot Var[g_x] + \sum_{x \in X} \sum_{y \in X \wedge y \neq x} a_x a_y \cdot Cov[g_x g_y]$$

Substituting from above, we finally get

$$\begin{aligned}
E[s(d, e)] &= L_d L_e \cdot \sum_{x \in X} a_x \cdot p_x^2 \\
Var[s(d, e)] &= L_d L_e \cdot \left[\sum_{x \in X} a_x p_x^2 + (L_d + L_e - 2) \sum_{x \in X} a_x p_x^3 - \right. \\
&\quad \left. (L_d + L_e - 1) \sum_{x \in X} a_x p_x^4 - (L_d + L_e - 1) \sum_{x \in X} \sum_{y \in X \wedge y \neq x} a_x a_y p_x^2 p_y^2 \right]
\end{aligned}$$

The summations in both $E[s(d, e)]$ and $Var[s(d, e)]$ can be precomputed, since they are the same for all document vector pairs.

For a Query Versus a Document Vector

The calculations above do not apply when we want to compare many item vectors against a fixed query vector q as in a search over a collection of text items. Our raw similarity measure becomes just

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

where query vector q has the components $\{q_x\}$, which do not have to be integer counts.

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

$$\begin{aligned}
E[s_q(d)] &= \sum_{x \in X} q_x \cdot E[f_x^d] \\
Var[s_q(d)] &= \sum_{x \in X} q_x^2 \cdot Var[f_x^d] + \sum_{x \in X} \sum_{y \in X \wedge y \neq x} q_x q_y \cdot Cov[f_x^d f_y^d]
\end{aligned}$$

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

$$\begin{aligned}
E[s_q(d)] &= L_d \cdot \left[\sum_{x \in X} q_x p_x \right] \\
Var[s_q(d)] &= L_d \cdot \left[\sum_{x \in X} q_x^2 p_x (1 - p_x) - \sum_{x \in X} \sum_{y \in X \wedge y \neq x} q_x q_y p_x p_y \right]
\end{aligned}$$

The distributional parameters for profile matching are much simpler to compute. The bracketed summations must be precomputed only once for each query, however. Except for automatic clustering, which must compare pairs of document vectors, similarity scaled with a simplified model for profile matching with fewer statistical degrees of freedom will become the workhorse of our text information processing with finite indexing.