Automatic Meaning Discovery Using Google

Rudi Cilibrasi[*]
CWI

Paul Vitanyi[†]
CWI, University of Amsterdam,
National ICT of Australia

Abstract

We have found a method to automatically extract the meaning of words and phrases from the world-wide-web using Google page counts. The approach is novel in its unrestricted problem domain, simplicity of implementation, and manifestly ontological underpinnings. The world-wide-web is the largest database on earth, and the latent semantic context information entered by millions of independent users averages out to provide automatic meaning of useful quality. We demonstrate positive correlations, evidencing an underlying semantic structure, in both numerical symbol notations and number-name words in a variety of natural languages and contexts. Next, we demonstrate the ability to distinguish between colors and numbers, and to distinguish between 17th century Dutch painters; the ability to understand electrical terms, religious terms, emergency incidents, and we conduct a massive experiment in understanding WordNet categories; the ability to do a simple automatic English-Spanish translation.

1 Introduction

Objects can be given literally, like the literal four-letter genome of a mouse, or the literal text of *War and Peace* by Tolstoy. For simplicity we take it that all meaning of the object is represented by the literal object itself. Objects can also be given by name, like "the four-letter genome of a mouse," or "the text of *War and Peace* by Tolstoy." There are also objects that cannot be given literally, but only by name and acquire their meaning from their contexts in background common knowledge in humankind, like "home" or "red." To make computers more intelligent one would like to represent meaning in computer-digestable form. Long-term and labor-intensive efforts like the *Cyc* project [14] and the *WordNet* project [23] try to establish semantic relations between common objects, or, more precisely, *names* for those objects. The idea is to create a semantic web of such vast proportions that rudimentary intelligence and knowledge about the real world spontaneously emerges. This comes at the great cost of designing structures capable of manipulating knowledge, and entering high quality contents in these structures by knowledgeable human experts. While the efforts are long-running and large scale, the overall information entered is minute compared to what is available on the world-wide-web.

The rise of the world-wide-web has enticed millions of users to type in trillions of characters to create billions of web pages of on average low quality contents. The sheer mass of the information available about almost every conceivable topic makes it likely that extremes will cancel and the majority or average is meaningful in a low-quality approximate sense. We devise a general method to tap the amorphous low-grade knowledge available for free on the world-wide-web, typed in by local users aiming at personal gratification of diverse objectives, and yet globally achieving what is effectively the largest semantic electronic database in the world. Moreover, this database is available for all by using any search engine that can return aggregate page-count estimates like Google for a large range of search-queries.

Previously, we and others developed a compression-based method to establish a universal similarity metric among objects given as finite binary strings [1, 17, 18, 5], which was widely reported [10, 11, 6]. Such objects can be genomes, music pieces in MIDI format, computer programs in Ruby or C, pictures in simple bitmap formats, or time sequences such as heart rhythm data. This method is feature-free in the sense that it doesn't analyze the files looking for particular features; rather it analyzes all features simultaneously and determines the similarity between every pair of objects according to the most dominant shared feature. The crucial point is that the method analyzes the objects themselves. This precludes comparison of abstract notions or other objects that don't lend themselves to direct analysis, like emotions, colors, Socrates, Plato, Mike Bonanno and Albert Einstein. While the previous method that compares the objects themselves is particularly suited to obtain knowledge about the similarity of objects themselves, irrespective of common beliefs about such similarities, here we develop a method that uses only the name of an object and obtains knowledge about the similarity of objects by tapping available information generated by multitudes of web users. Here we are reminded of the words of D.H. Rumsfeld [21]

> "A trained ape can know an awful lot
> Of what is going on in this world,
> Just by punching on his mouse
> For a relatively modest cost!"

This is useful to extract knowledge from a given corpus of knowledge, in this case the Google database, but not to obtain true facts that are not common knowledge in that database. For example, common viewpoints on the creation myths in different religions may be extracted by the Googling method, but contentious questions of fact concerning the phylogeny of species can be better approached by using the genomes of these species, rather than by opinion.

1.1 Googling for Knowledge

Intuitively, the approach is as follows. The Google search engine indexes around ten billion pages on the web today. Each such page can be viewed as a set of index terms. A search for a particular index term, say "horse", returns a certain number of hits (web pages where this term occurred), say 46,700,000. The number of hits for the search term "rider" is, say, 12,200,000. It is also possible to search for the pages where both "horse" and "rider" occur. This gives, say, 2,630,000 hits. This can be easily put in the standard probabilistic framework. If w is a web page and x a search term, then we write $x \in w$ to mean that Google returns web page w when presented with search term x. An *event* is a set of web pages returned by Google after it has been presented by a search term. We can view the event as the collection of all contexts of the search term, background knowledge, as induced by the accessible web pages for the Google search engine. If the search term is x, then we denote the event by $\mathbf{x}$, and define $\mathbf{x} = \{w : x \in w\}$. The *probability* $p(x)$ of an event $\mathbf{x}$ is the number of web pages in the event divided by the overall number M of web pages possibly returned by Google. Thus, $p(x) = |\mathbf{x}|/M$. At the time of writing, Google searches 8,058,044,651 web pages. Define the joint event $\mathbf{x} \bigcap \mathbf{y} = \{w : x, y \in w\}$ as the set of web pages returned by Google, containing both the search term x and the search term y. The joint probability $p(x,y) = |\{w : x, y \in w\}|/M$ is the number of web pages in the joint event divided by the overall number M of web pages possibly returned by Google. This notation also allows us to define the probability $p(x|y)$ of *conditional* events $\mathbf{x}|\mathbf{y} = (\mathbf{x} \bigcap \mathbf{y})/\mathbf{y}$ defined by $p(x|y) = p(x,y)/p(y)$.

In the above example we have therefore $p(horse) \approx 0.0058$, $p(rider) \approx 0.0015$, $p(horse, rider) \approx 0.0003$. We conclude that the probability $p(horse|rider)$ of "horse" accompanying "rider" is $\approx 1/5$ and the probability $p(rider|horse)$ of "rider" accompanying "horse" is $\approx 1/19$. The probabilities are asymmetric, and it is the least probability that is the significant one. A very general search term like "the" occurs in virtually all (English language) web pages. Hence $p(the|rider) \approx 1$, and for almost all search terms x we have $p(the|x) \approx 1$. But $p(rider|the) \ll 1$, say about equal to $p(rider)$, and gives the relevant information about the association of the two terms.

Our first attempt is therefore a distance

$$D_1(x,y) = \min\{p(x|y), p(y|x)\}.$$

Experimenting with this distance gives bad results. One reason being that the differences among small probabilities have increasing significance the smaller the probabilities involved are. Another reason is that we deal with absolute probabilities: two notions that have very small probabilities each and have D_1-distance ε are much less similar than

two notions that have much larger probabilities and have the same D_1-distance. To resolve the first problem we take the negative logarithm of the items being minimized, resulting in

$$D_2(x,y) = \max\{\log 1/p(x|y), \log 1/p(y|x)\}.$$

To resolve the second problem we normalize $D_2(x,y)$ by dividing by the maximum of $\log 1/p(x), \log 1/p(y)$. Altogether, we obtain the following normalized distance

$$D_3(x,y) = \frac{\max\{\log 1/p(x|y), \log 1/p(y|x)\}}{\max\{\log 1/p(x), \log 1/p(y)\}},$$

for $p(x|y) > 0$ (and hence $p(y|x) > 0$), and $D_3(x,y) = \infty$ for $p(x|y) = 0$ (and hence $p(y|x) = 0$). Note that $p(x|y) = p(x,y)/p(x) = 0$ means that the search terms "x" and "y" never occur together. The two conditional complexities are either both 0 or they are both strictly positive. Moreover, if either of $p(x), p(y)$ is 0, then so are the conditional probabilities, but not necessarily vice versa.

We note that in the conditional probabilities the total number M, of web pages indexed by Google, is divided out. Therefore, the conditional probabilities are independent of M, and can be replaced by the number of pages, the *frequency*, returned by Google. Define the *frequency* $f(x)$ of search term x as the number of pages a Google search for x returns: $f(x) = Mp(x)$, $f(x,y) = Mp(x,y)$, and $p(x|y) = f(x,y)/f(y)$. Rewriting D_3 results in our final notion, the *normalized Google distance (NGD)*, defined by

$$\text{NGD}(x,y) = \frac{\max\{\log f(x), \log f(y)\} - \log f(x,y)}{\log M - \max\{\log f(x), \log f(y)\}}, \tag{1}$$

and if $f(x), f(y) > 0$ and $f(x,y) = 0$ then $\text{NGD}(x,y) = \infty$. From (1) we see that

1. $\text{NGD}(x,y)$ is undefined for $f(x) = f(y) = 0$;

2. $\text{NGD}(x,y) = \infty$ for $f(x,y) = 0$ and either or both $f(x) > 0$ and $f(y) > 0$; and

3. $\text{NGD}(x,y) \geq 0$ otherwise.

REMARK 1 Note that $\text{NGD}(x,y)$ can exceed 1 for natural search terms. For example, the two terms: "by", generating a count of 2,770,000,000, and "with", generating a count of 2,566,000,000, together give a count of only 49,700,000. Using (1) (with $M = 8,058,044,651$), we obtain $\text{NGD}(by, with) = \approx 1.43$. The explanation is that two terms that just happen to never occur together for whatever reason, but do occur separately often, can create an NGD that is high. We can go even higher using terms "the", count 8,000,000,000, and "szyzygy", count 175, with count 161 for the joint pair. This gives an NGD of over 1000.

In practice, however, most pairs' NGD 's come between 0 and 1. This is not due to an imperfection in the method. Rather, NGD values greater than 1 may be thought to correspond to the idea of *negative correlation* in probability theory; we may venture to guess, for instance, that the English words "by" and "with" are both very popular, yet can often be used as substitutes for one another, and thus a rather large proportion of webpages contain either "by" or "with" but not both. In the case of NID, (3), we meet later, the subadditivity property of K yields the upper bound of 1, but there is no such restriction for an arbitrary probability distribution such as Google. ◊

REMARK 2 With the Google hit numbers above, we can now compute

$$\text{NGD}(horse, rider) \approx 0.453.$$

We did the same calculation when Google indexed only one-half of the current number of pages: 4,285,199,774. It is instructive that the probabilities of the used search terms didn't change significantly over this doubling of pages, with number of hits for "horse" equal 23,700,000, for "rider" equal 6,270,000, and for "horse, rider" equal to 1,180,000. The $\text{NGD}(horse, rider)$ we computed in that situation was 0.4445. This is in line with our contention that the relative frequencies of web pages containing search terms gives objective information about the semantic relations between the search terms. If this is the case, then with the vastness of the information accessed by Google the Google probabilities of search terms and the computed NGD 's should stabilize (be scale invariant) with a growing Google database. ◊

3

The above derivation gives the intuition. However, the original thinking which led to the NGD derived from a mathematical theory of similarity based on Kolmogorov complexity, and we shall trace this derivation in Section 2. The following are the main properties of the NGD :

1. The *range* of the NGD is in between 0 and ∞;

   (a) If $x = y$ or if $x \neq y$ but frequency $M > f(x) = f(y) = f(x,y) > 0$, then $NGD(x,y) = 0$. That is, the semantics of x and y in the Google sense is the same.

   (b) If frequency $f(x) = 0$, then for every search term y we have $f(x,y) = 0$, and the $NGD(x,y) = \log f(y)/\log(M/f(y))$.

2. The NGD is always nonnegative and $NGD(x,x) = 0$ for every x. For every pair x,y we have $NGD(x,y) = NGD(y,x)$: it is symmetric. However, the NGD is *not a metric*: it does not satisfy $NGD(x,y) > 0$ for every $x \neq y$. For example, choose $x \neq y$ with $\mathbf{x} = \mathbf{y}$. Then, $f(x) = f(y) = f(x,y)$ and $NGD(x,y) = 0$. Nor does the NGD satisfy the triangle inequality $NGD(x,y) \leq NGD(x,z) + NGD(z,y)$ for all x,y,z. For example, choose $\mathbf{z} = \mathbf{x} \bigcup \mathbf{y}$, $\mathbf{x} \bigcap \mathbf{y} = \emptyset$, $\mathbf{x} = \mathbf{x} \bigcap \mathbf{z}$, $\mathbf{y} = \mathbf{y} \bigcap \mathbf{z}$, and $|\mathbf{x}| = |\mathbf{y}| = \sqrt{M}$. Then, $f(x) = f(y) = f(x,z) = f(y,z) = \sqrt{M}$, $f(z) = 2\sqrt{M}$, and $f(x,y) = 0$. This yields $NGD(x,y) = 1$ and $NGD(x,z) = NGD(z,y) = 2/\log(M/4)$, which violates the triangle inequality for $M > 64$.

3. The NGD is *scale-invariant*. It is very important that, if the number M of pages indexed by Google grows sufficiently large, the number of pages containing given search terms goes to a fixed fraction of M, and so does the number of pages containing conjunctions of search terms. This means that if M doubles, then so do the f-frequencies. For the NGD to give us an objective semantic relation between search terms, it needs to become stable when the number M of indexed pages grows. Some evidence that this actually happens is given in Remark 2.

REMARK 3 If frequency $f(x) = M$ and $f(x,y) < f(x)$, then $NGD(x,y) = \infty$. If $f(x) = f(x,y) = M$, then $NGD(x,y) = 0/0$, that is, it is undefined. These anomalies, and related anomalies, are redressed by the proper formulation (7) of the NGD , with quantity $N > M$ replacing M, in the theoretical analysis of Section 3.2. $\diamond$

1.2 Outline

We first start with a technical introduction outlining both the classical information theoretic notions underpinning our approach, as well as the more novel ideas of Kolmogorov complexity, information distance, and compression-based similarity metric (Section 2). Then we give a technical description of the Google distribution, the Normalized Google Distance, and the universality of these notions (Section 3.2), ended by Subsection 3.1 pressing home the difference between literal-object based similarity (as in compression-based similarity), and context-based similarity between objects that are not given literally but only in the form of names that acquire their meaning through contexts in databases of background information (as in Google-based similarity). In Section 4 we present a plethora of clustering and classification experiments to validate the universality, robustness, and accuracy of our proposal. A mass of experimental work, which for space reasons can not be reported here, is available [3]. In Appendix A we explain some basics of the SVM approach we use in the classification experiments, where the Google similarity is used to extract feature vectors used by the kernel.

2 Technical Preliminaries

For convenience we restrict ourselves to binary source words and binary code words—every finite alphabet can be recoded in binary in a way that is theory neutral. If $x \in \{0,1\}^*$, then $|x|$ denotes the *length* (number of bits) of x. Let ε denote the *empty* word, that is, $|\varepsilon| = 0$. A binary string y is a *proper prefix* of a binary string x if we can write $x = yz$ for $z \neq \varepsilon$. A set $\{x,y,\ldots\} \subseteq \{0,1\}^*$ is *prefix-free* if no element is a proper prefix of any other. A prefix-free set is also called a *prefix code* and its elements are called *code words*. An example of a prefix code, that is used later, encodes a source word $x = x_1 x_2 \ldots x_n$ by the code word

$$\bar{x} = 1^n 0 x.$$

4

This prefix-free code is called *self-delimiting*, because there is a fixed computer program associated with this code that can determine where the code word $\bar{x}$ ends by reading it from left to right without backing up. This way a composite code message can be parsed in its constituent code words in one pass by a computer program. (This desirable property holds for every prefix-free encoding of a finite set of source words, but not for every prefix-free encoding of an infinite set of source words. For a single finite computer program to be able to parse a code message the encoding needs to have a certain uniformity property like the $\bar{x}$ code.)

2.1 Prefix codes and the Kraft inequality

Let X be the set of natural numbers and consider the straightforward non-prefix binary representation with the ith binary string in the length-increasing lexicographical order corresponding to the number i. There are two elements of X with a description of length 1, four with a description of length 2 and so on. However, there are less binary prefix code words of each length: if x is a prefix code word then no $y = xz$ with $z \neq \varepsilon$ is a prefix code word. Asymptotically there are less prefix code words of length n than the 2^n source words of length n. Quantification of this intuition for countable X and arbitrary prefix-codes leads to a precise constraint on the number of code-words of given lengths. This important relation is known as the *Kraft Inequality* and is due to L.G. Kraft [8].

LEMMA 1 *Let $l_1, l_2, \ldots$ be a finite or infinite sequence of natural numbers. There is a prefix-code with this sequence as lengths of its binary code words iff*

$$\sum_n 2^{-l_n} \leq 1. \tag{2}$$

2.2 Uniquely Decodable Codes

We want to code elements of X in a way that they can be uniquely reconstructed from the encoding. Such codes are called *uniquely decodable*. Every prefix-code is a uniquely decodable code. On the other hand, not every uniquely decodable code satisfies the prefix condition. Prefix-codes are distinguished from other uniquely decodable codes by the property that the end of a code word is always recognizable as such. This means that decoding can be accomplished without the delay of observing subsequent code words, which is why prefix-codes are also called instantaneous codes. There is good reason for our emphasis on prefix-codes. Namely, it turns out that Lemma 1 stays valid if we replace "prefix-code" by "uniquely decodable code." This important fact means that every uniquely decodable code can be replaced by a prefix-code without changing the set of code-word lengths.

2.3 Probability distributions and complete prefix codes

A uniquely decodable code is *complete* if the addition of any new code word to its code word set results in a non-uniquely decodable code. It is easy to see that a code is complete iff equality holds in the associated Kraft Inequality. Let $l_1, l_2, \ldots$ be the code words of some complete uniquely decodable code. Let us define $q_x = 2^{-l_x}$. By definition of completeness, we have $\sum_x q_x = 1$. Thus, the q_x can be thought of as *probability mass functions* corresponding to some probability distribution Q. We say Q is the distribution *corresponding* to $l_1, l_2, \ldots$. In this way, each complete uniquely decodable code is mapped to a unique probability distribution. Of course, this is nothing more than a formal correspondence: we may choose to encode outcomes of X using a code corresponding to a distribution q, whereas the outcomes are actually distributed according to some $p \neq q$. But, as we argue below, if X is distributed according to p, then the code to which p corresponds is, in an average sense, the code that achieves optimal compression of X. In particular, every probability mass function p is related to a prefix code, the *Shannon-Fano code*, such that the expected number of bits per transmitted code word is as low as is possible for any prefix code, assuming that a random source X generates the source words x according to $P(X = x) = p(x)$. The Shannon-Fano prefix code encodes a source word x by a code word of length $l_x = \lceil \log 1/x \rceil$, so that the expected transmitted code word length equals $\sum_x p(x) \log 1/p(x) = H(X)$, the entropy of the source X, up to one bit. This is optimal by Shannon's "noiseless coding" theorem [22].

5

2.4 Kolmogorov Complexity

The basis of much of the theory explored in this paper is Kolmogorov complexity. For an introduction and details see the textbook [19]. Here we give some intuition and notation. We assume a fixed reference universal programming system. Such a system may be a general computer language like LISP or Ruby, and it may also be a fixed reference universal Turing machine in a given standard enumeration of Turing machines. The latter choice has the advantage of being formally simple and hence easy to theoretically manipulate. But the choice makes no difference in principle, and the theory is invariant under changes among the universal programming systems, provided we stick to a particular choice. We only consider universal programming systems such that the associated set of programs is a prefix code— as is the case in all standard computer languages. The *Kolmogorov complexity* of a string x is the length, in bits, of the shortest computer program of the fixed reference computing system that produces x as output. The choice of computing system changes the value of $K(x)$ by at most an additive fixed constant. Since $K(x)$ goes to infinity with x, this additive fixed constant is an ignorable quantity if we consider large x.

One way to think about the Kolmogorov complexity $K(x)$ is to view it as the length, in bits, of the ultimate compressed version from which x can be recovered by a general decompression program. Compressing x using the compressor *gzip* results in a file x_g with (for files that contain redundancies) the length $|x_g| < |x|$. Using a better compressor *bzip2* results in a file x_b with (for redundant files) usually $|x_b| < |x_g|$; using a still better compressor like *PPMZ* results in a file x_p with (for again appropriately redundant files) $|x_p| < |x_b|$. The Kolmogorov complexity $K(x)$ gives a lower bound on the ultimate value: for every existing compressor, or compressors that are possible but not known, we have that $K(x)$ is less or equal to the length of the compressed version of x. That is, $K(x)$ gives us the ultimate value of the length of a compressed version of x (more precisely, from which version x can be reconstructed by a general purpose decompresser), and our task in designing better and better compressors is to approach this lower bound as closely as possible.

2.5 Normalized Information Distance

In [1] we considered a related notion: given two strings x and y, what is the length of the shortest binary program in the reference universal computing system such that the program computes output y from input x, and also output x from input y. This is called the *information distance* and denoted as $E(x,y)$. It turns out that, up to a negligible logarithmic additive term,

$$E(x,y) = K(x,y) - \min\{K(x), K(y)\}.$$

This distance $E(x,y)$ is actually a metric: up to close precision we have $E(x,x) = 0$, $E(x,y) > 0$ for $x \neq y$, $E(x,y) = E(y,x)$ and $E(x,y) \leq E(x,z) + E(z,y)$, for all x,y,z. We now consider a large class of *admissible distances*: all distances (not necessarily metric) that are nonnegative, symmetric, and *computable* in the sense that for every such distance D there is a prefix program that, given two strings x and y, has binary length equal to the distance $D(x,y)$ between x and y. Then,

$$E(x,y) \leq D(x,y) + c_D,$$

where c_D is a constant that depends only on D but not on x,y. This justifies calling the information distance E "universal"; it minorizes every computable distance. If two strings x and y are close according to *some* computable distance D, then they are at least as close according to distance E. Since every feature in which we can compare two strings can be quantified in terms of a distance, and every distance can be viewed as expressing a quantification of how much of a particular feature the strings have in common (the feature being quantified by that distance), the information distance determines the distance between two strings according to the dominant feature in which they are similar. If small strings differ by an information distance which is large compared to their sizes, then the strings are very different. However, if two very large strings differ by the same (now relatively small) information distance, then they are very similar. Therefore, the information distance itself is not suitable to express true similarity. For that we must define a relative information distance: we need to normalize the information distance. Such an approach was first proposed in [17] in the context of genomics-based phylogeny, and improved in [18] to the one we use here. The *normalized information distance (NID)* has values between 0 and 1, and it inherits the universality of the information distance in the sense that it in the appropriate sense minorizes every other possible normalized computable distance (suitably defined). In the same way as before we can identify the computable normalized distances with computable

similarities according to some features, and the NID discovers for every pair of strings the feature in which they are most similar, and expresses that similarity on a scale from 0 to 1 (0 being the same and 1 being completely different in the sense of sharing no features). The NID is defined by

$$\text{NID}(x,y) = \frac{K(x,y) - \min(K(x),K(y))}{\max(K(x),K(y))}. \tag{3}$$

It has several wonderful properties that justify its description as the most informative metric [18].

2.6 Normalized Compression Distance

Unfortunately, the NID is uncomputable since the constituent K is. But we can use real data compression programs to approximate K: a compression algorithm defines a computable function and hence the number of bits of the compressed version of a string is an upper bound on Kolmogorov complexity of that string, up to an additive constant depending on the compressor but not on the string in question. Thus, if C is a compressor and we use $C(x)$ to denote the length of the compressed version of a string x, then we arrive at the *Normalized Compression Distance*:

$$\text{NCD}(x,y) = \frac{C(xy) - \min(C(x),C(y))}{\max(C(x),C(y))}, \tag{4}$$

where for convenience we have replaced the pair (x,y) in the formula by the concatenation xy. This transition raises several tricky problems, for example how the NCD approximates the NID if C approximates K, see [5], which do not need to concern us here. Thus, the NCD is actually a family of compression functions parameterized by the given data compressor C. The NID is the limiting case, where $K(x)$ denotes the number of bits in the shortest code for x from which x can be decompressed by a general purpose computable decompressor.

3 Feature-Free Extraction of Semantic Relations with Google

The number of web pages currently indexed by Google is approaching 10^{10}. Every common search term occurs in millions of web pages. This number is so vast, and the number of web authors generating web pages is so enormous (and can be assumed to be a truly representative very large sample from humankind), that the relations represented by the Normalized Google Distance (1) approximately capture the assumed true semantic relations governing the search terms. Determining the NGD between two Google search terms does not involve analysis of particular features or specific background knowledge of the problem area. Instead, it analyzes all features automatically through Google searches of the most general background knowledge data base: the world-wide-web. (In Statistics "parameter-free estimation" means that the number of parameters analyzed is infinite or not a priori restricted. In our setting "feature-freeness" means that we analyze all features.)

3.1 Genesis of the Approach

We start from the observation that a compressor defines a code word length for every source word, namely, the number of bits in the compressed version of that source word. Viewing this code as a Shannon-Fano code, Section 2.3, it defines in its turn a probability mass function on the source words. Conversely, every probability mass function of the source words defines a Shannon-Fano code of the source words. Since this code is optimally compact in the sense of having expected code-word length equal to the entropy of the initial probability mass function, we take the viewpoint that a probability mass function is a compressor.

EXAMPLE 1 For example, the NID uses the probability mass function $\mathbf{m}(x) = 2^{-K(x)}$. This function is not computable, but it has the weaker property of being lower semi-computable: by approximating $K(x)$ from above by better and better compressors, we approximate $\mathbf{m}(x)$ from below. The distribution $\mathbf{m}(x)$ has the remarkable property that it dominates every lower semi-computable probability mass function $P(x)$ (and hence all computable ones) by assigning more probability to every finite binary string x than $P(x)$, up to a multiplicative constant $c_P > 0$ depending on P but not on x ($\mathbf{m}(x) \geq c_P P(x)$). We say that $\mathbf{m}(x)$ is *universal* for the enumeration of all lower semi-computable probability

7

mass functions, [19], a terminology that is closely related to the "universality" of a universal Turing machine in the enumeration of all Turing machines. It is this property that allows us to show [18] that NID is the *most informative* distance (actually a metric) among a large family of (possibly non-metric) "admissible normalized information distances." But we cannot apply these same formal claims to the real-world NCD , except in a sense that is relativized on how well the compressor approximates the ultimate Kolmogorov complexity [5, 4]. In these papers, we have investigated NCD in a wide variety of domains including genomics, language evolution, literature, optical character recognition, music recognition, and time sequences. Our experiments show that NCD , using a range of different compressors, consistently yield significant results in these contexts. ◇

In essence, the choice of compressor brings a particular set of assumptions to bear upon an incoming data stream, and if these assumptions turn out to be accurate for the data in question, then compression is achieved. This is the same as saying that the probability mass function defined by the compressor concentrates high probability on these data. If a pair of files share information in a way that matches the assumptions of a particular compressor, then we obtain a low NCD . Every compressor analyzes the string to be compressed by quantifying an associated family of features. A compressor such as gzip detects a class of features, for example matching substrings that are separated by no more than 32 kilobytes. Certain higher-order similarities are detected in the final Huffman coding phase. This explains how gzip is able to correctly cluster files generated by Bernoulli processes. A better compressor like bzip2 detects substring matches across a wider window of 900 kilobytes, and detects an expanded set of higher-order features. Such compressors implicitly assume that the data has no global, structured, meaning. The compressor only looks for statistical biases, repetitions, and other biases in symmetrically defined local contexts, and cannot achieve compression even for very low-complexity meaningful strings like the digits of π. They assume the data source is at some level a simple stationary ergodic random information source which is by definition meaningless. The problem with this is clearly sketched by the great probabilist A.N. Kolmogorov [15, 16]: "The probabilistic approach is natural in the theory of information transmission over communication channels carrying 'bulk' information consisting of a large number of unrelated or weakly related messages obeying definite probabilistic laws. ... [it] can be convincingly applied to the information contained, for example, in a stream of congratulatory telegrams. But what real meaning is there, for example, in [applying this approach to] 'War and Peace'? · Or, on the other hand, must we assume that the individual scenes in this book form a random sequence with 'stochastic relations' that damp out quite rapidly over a distance of several pages?" The compressors apply no external knowledge to the compression, and so will not take advantage of the fact that U always follows Q in the English language, and instead must learn this fact anew for each separate file (or pair) despite the simple ubiquity of this rule. Thus the generality of common data compressors is also a liability, because the features which are extracted are by construction meaningless and devoid of relevance.

Yet, files exist in the real world, and the files that actually exist in stored format by and large carry a tremendous amount of structural, global, meaning; if they didn't then we would throw them away as useless. They do exhibit heavy biases in terms of the meaningless features, for instance in the way the letters T and E occur more frequently in English than Z or Q, but even this fails to capture the heart of the reason of the file's existence in the first place: because of its relevance to the rest of the world. But gzip doesn't know this reason; it is as if everywhere gzip looks it only finds a loaded die or biased coin, resolute in its objective and foolish consistency. In order to address this coding deficiency we choose an opposing strategy: instead of trying to apply no external knowledge at all during compression, we try to apply as much as we can from as many sources as possible simultaneously, and in so doing attempt to capture not the *literal* part but instead the *contextualized importance* of each string within a greater and all-inclusive whole.

Thus, instead of starting with a standard data compression program, we start from a probability mass function that reflects knowledge, and construct the corresponding Shannon-Fano code to convert probabilities to code word lengths, and apply the NCD formula. At this moment one database stands out as the pinnacle of computer-accessible human knowledge and the most inclusive summary of statistical information: the Google search engine. There can be no doubt that Google has already enabled science to accelerate tremendously and revolutionized the research process. It has dominated the attention of internet users for years, and has recently attracted substantial attention of many Wall Street investors, even reshaping their ideas of company financing. We have devised a way to interface the Google search engine to our NCD software to create a new type of pseudo-compressor based NCD , and call this new distance the Normalized Google Distance, or NGD . We have replaced the obstinate objectivity of classical compressors with an anthropomorphic subjectivity derived from the efforts of millions of people worldwide. Experiments suggest that this new distance shares some strengths and weaknesses in common with the humans that have helped create it: it is highly

adaptable and nearly unrestricted in terms of domain, but at the same time is imprecise and fickle in its behavior. It is limited in that it doesn't analyze the literal objects like the NCD does, but instead uses names for those objects in the form of ASCII search terms or tuples of terms as inputs to extract the meaning of those objects from the total of information on the world-wide-web.

EXAMPLE 2 An example may help clarify the distinction between these two opposing paradigms. Consider the following sequence of letters:

U Q B

Assume that the next letter will be a vowel. What vowel would you guess is most likely, in the absence of any more specific information? One common assumption is that the samples are i.i.d. (identical, independently distributed), and given this assumption a good guess is U ; since it has already been shown once, chances are good that U is weighted heavily in the true generating distribution. In assuming i.i.d.-ness, we implicitly assume that there is some true underlying random information source. This assumption is often wrong in practice, even in an approximate sense. Changing the problem slightly, using English words as tokens instead of just letters, suppose we are given the sequence

the quick brown

Now we are told that the next word has three letters and does not end the sentence. We may imagine various three letter words that fit the bill. On an analysis as before, we ought to expect the to continue the sentence. The computer lists 535 English words of exactly three letters. We may use the gzip data compressor to compute the NCD for each possible completion like this: NCD (the quick brown,cow) , NCD (the quick brown,the) , and so on, for all of the 3-letter words. We may then sort the words in ascending order of NCD and this yields the following words in front, all with NCD of 0.61: own, row, she, the . There are other three letter words, like hot, that have NCD of 0.65, and many with larger distance. With such very small input strings, there are granularity effects associated with the compressor rounding to full bytes, which makes compression resolve only to the level of 8 bits at best. So as we might expect, gzip is using a sort of inference substantially similar to the sort that might lead the reader to guess U as a possible completion in the first example above.

Consider now what would happen if we were to use Google instead of gzip as the data compressor. Here we may change the input domain slightly; before, we operated on general strings, binary or otherwise. With Google, we will restrict ourselves to ASCII words that can be used as search terms in the Google Search engine. With each search result, Google returns a count of matched pages. This can be thought to define a function mapping search terms (or combinations thereof) to page counts. This, in turn, can be thought to define a probability distribution, and we may use a Shannon-Fano code to associate a code length with each page count. We divide the total number of pages returned on a query by the maximum that can be returned, when converting a page count to a probability; at the time of these experiments, the maximum was about 5,000,000,000. Computing the NGD of the phrase the quick brown, with each three-letter word that may continue the phrase (ignoring the constraint that that word immediately follows the word brown), we arrive at these first five most likely (candidate, NGD)-pairs (using the Google values at the time of writing):

```
fox: 0.532154812757325
vex: 0.561640307093518
jot: 0.579817813761161
hex: 0.589457285818459
pea: 0.604444512168738
```

As many typing students no doubt remember, a popular phrase to learn the alphabet is *The quick brown fox jumped over the lazy dog.* It is valuable because it uses every letter of the English alphabet.   ◇

Thus, we see a contrast between two styles of induction: The first type is the NCD based on a *literal* interpretation of the data: the data is the object itself. The second type is the NGD based on interpreting the data as a *name* for an abstract object which acquires its meaning from masses of *contexts* expressing a large body of common-sense knowledge. It may be said that the first case ignores the meaning of the message, whereas the second focuses on it.

9

3.2 The Google Distribution

Using the Google search engine we obtain the number of web pages containing the requested search term, as in Section 1. That is what we actually use in our experiments. For our theoretical analysis, where we want to express the Shannon-Fano code length of encoding a search term based on a probability of the associated event, this will turn out not proper. Instead, we need a minor renormalization, which in typical situations only slightly changes the values according to (1), and resolves the anomalies observed in Remark 3.

Let the set of singleton *Google search terms* be denoted by $\mathcal{S}$. In the sequel we use both singleton search terms and doubleton search terms $\{\{x,y\} : x,y \in \mathcal{S}\}$. Let the set of web pages indexed (possible of being returned) by Google be Ω. The cardinality of Ω is denoted by $M = |\Omega|$, and currently $8 \cdot 10^9 \le M \le 9 \cdot 10^9$. Assume that a priori all web pages are equi-probable, with the probability of being returned by Google being $1/M$. A subset of Ω is called an *event*. Every *search term* x usable by Google defines a *singleton Google event* $\mathbf{x} \subseteq \Omega$ of web pages that contain an occurrence of x and are returned by Google if we do a search for x. Let $L : \Omega \to [0,1]$ be the uniform mass probability function. The probability of such an event $\mathbf{x}$ is $L(\mathbf{x}) = |\mathbf{x}|/M$. Similarly, the *doubleton Google event* $\mathbf{x} \cap \mathbf{y} \subseteq \Omega$ is the set of web pages returned by Google if we do a search for pages containing both search term x and search term y. The probability of this event is $L(\mathbf{x} \cap \mathbf{y}) = |\mathbf{x} \cap \mathbf{y}|/M$. We can also define the other Boolean combinations: $\neg \mathbf{x} = \Omega \backslash \mathbf{x}$ and $\mathbf{x} \bigcup \mathbf{y} = \Omega \backslash (\neg \mathbf{x} \cap \neg \mathbf{y})$, each such event having a probability equal to its cardinality divided by M. If $\mathbf{e}$ is an event obtained from the basic events $\mathbf{x}, \mathbf{y}, \ldots$, corresponding to basic search terms $x, y, \ldots$, by finitely many applications of the Boolean operations, then the probability $L(\mathbf{e}) = |\mathbf{e}|/M$.

Google events capture in a particular sense all background knowledge about the search terms concerned available (to Google) on the web. Therefore, it is natural to consider code words for those events as coding this background knowledge. However, we cannot use the probability of the events directly to determine a prefix code such as the Shannon-Fano code as in Section 2.3. The reason is that the events overlap and hence the summed probability exceeds 1. By the Kraft inequality (2) this prevents a corresponding Shannon-Fano code. The solution is to normalize: We use the probability of the Google events to define a probability mass function over the set $\{\{x,y\} : x,y \in \mathcal{S}\}$ of Google search terms, both singleton and doubleton. Define

$$N = \sum_{\{x,y\} \subseteq \mathcal{S}} |\mathbf{x} \cap \mathbf{y}|,$$

counting each singleton set and each doubleton set (by definition unordered) once in the summation. Since every web page that is indexed by Google contains at least one occurrence of a search term, we have $N \ge M$. On the other hand, web pages contain on average not more than a certain constant α search terms. Therefore, $N \le \alpha M$. Define

$$g(x) = L(\mathbf{x})M/N = |\mathbf{x}|/N \tag{5}$$
$$g(x,y) = L(\mathbf{x} \cap \mathbf{y})M/N = |\mathbf{x} \cap \mathbf{y}|/N.$$

Then, $\sum_{x \in \mathcal{S}} g(x) + \sum_{x,y \in \mathcal{S}} g(x,y) = 1$. Note that $g(x,y)$ is not a conventional joint distribution since possibly $g(x) \ne \sum_{y \in \mathcal{S}} g(x,y)$. Rather, we consider g to be a probability mass function over the sample space $\{\{x,y\} : x,y \in \mathcal{S}\}$. This g-distribution changes over time, and between different samplings from the distribution. But let us imagine that g holds in the sense of an instantaneous snapshot. The real situation will be an approximation of this. Given the Google machinery, these are absolute probabilities which allow us to define the associated Shannon-Fano code for both the singletons and the doubletons. The *Google code* G is defined by

$$G(x) = \log 1/g(x) \tag{6}$$
$$G(x,y) = \log 1/g(x,y).$$

In contrast to strings x where the complexity $C(x)$ represents the length of the compressed version of x using compressor C, for a search term x (just the name for an object rather than the object itself), the Google code of length $G(x)$ represents the shortest expected prefix-code word length of the associated Google event $\mathbf{x}$. The expectation is taken over the Google distribution p. In this sense we can use the Google distribution as a compressor for Google "meaning" associated with the search terms. The associated NCD , now called the *normalized Google distance (NGD)* is then

defined by (1) with N substituted for M, rewritten as

$$\text{NGD}(x,y) = \frac{G(x,y) - \min(G(x),G(y))}{\max(G(x),G(y))}. \tag{7}$$

This NGD is an approximation to the NID of (3) using the Shannon-Fano code (Google code) generated by the Google distribution as defining a compressor approximating the length of the Kolmogorov code, using the background knowledge on the web as viewed by Google as conditional information.

LEMMA 2 *The* NGD *according to* (7) *equals the practically used* NGD *according to* (1) *up to a multiplicative factor*

$$\delta(x,y) = \frac{\log M - \max\{\log f(x), \log f(y)\}}{\log N - \max\{\log f(x), \log f(y)\}}.$$

PROOF. Straightforward rewriting. □

REMARK 4 This factor $0 \le \delta(x,y) \le 1$ is the price we pay for proper prefix coding of the Google events corresponding to the search terms. We assume that $M \le N \le 1000M$, since on average web pages probably do not contain more than 1000 Google search terms. Therefore, $\log N - \log M \le 10$. Used in practice, (7) may possibly improve the results in case $\max\{\log f(x), f(y)\}$ is large, say 20. With $M = 5 \cdot 10^9$, and $N = 10^3 M$, we have $\delta(x,y) \approx 1/2$. But for $\max\{\log f(x), f(y)\}$ is small, say 10, we have $\delta(x,y) \approx 2/3$. Indeed, this normalization (7) using N instead of M also resolves the anomalies with (1) discussed in Remark 3. In fact, however, our experimental results suggest that every reasonable (greater than any $f(x)$) value can be used for the normalizing factor N, and our results seem in general insensitive to this choice. In our software, this parameter N can be adjusted as appropriate, and we often use M for N. ◊

REMARK 5 Like the NGD according to (1), the one according to (7) is nonnegative for all x,y, $\text{NGD}(x,x) = 0$ for every x, and it is symmetric. But again it is not a metric since there are x,y, $x \ne y$, with $\text{NGD}(x,y) = 0$, and the triangle inequality is violated. The example used to show violation of the triangle inequality of (1) yields here: $\text{NGD}(x,y) = \frac{1}{2}\log M/(\log N - \frac{1}{2}\log M)$ and $\text{NGD}(x,z) = \text{NGD}(z,y) = 1/(\log N - \frac{1}{2}\log M - 1)$. ◊

3.3 Universality of Google Distribution

A central notion in the application of compression to learning is the notion of "universal distribution," see [19]. Consider an effective enumeration $\mathcal{P} = p_1, p_2, \ldots$ of probability mass functions with domain $\mathcal{S}$. The list $\mathcal{P}$ can be finite or countably infinite.

DEFINITION 1 A probability mass function p_u occurring in $\mathcal{P}$ is *universal* for $\mathcal{P}$, if for every p_i in $\mathcal{P}$ there is a constant $c_i > 0$ and $\sum_{i \ne u} c_i \ge 1$, such that for every $x \in \mathcal{S}$ we have $p_u(x) \ge c_i \cdot p_i(x)$. Here c_i may depend on the indexes u, i, but not on the functional mappings of the elements of list $\mathcal{P}$ nor on x.

If p_u is universal for $\mathcal{P}$, then it immediately follows that for every p_i in $\mathcal{P}$, the Shannon-Fano code-word length for source word x, Section 2.3, associated with p_u, minorizes the Shannon-Fano code-word length associated with p_i, by satisfying $\log 1/p_u(x) \le \log 1/p_i(x) + \log c_i$, for every $x \in \mathcal{S}$.

Now consider the particular (finite) list of probability mass functions of search terms, each probability mass function associated with its web author in the list $\mathcal{A} = 1, 2, \ldots, a$ of *web authors producing pages* on the web. In the following we simply model the generated probability mass functions via frequency analysis, and make no claims as to how the pages were made. For convenience we assume that each page is generated by a single web author, and that the pages are indexed by Google. Let web author i of the list $\mathcal{A}$ produce the set of web pages Ω_i and denote $M_i = |\Omega_i|$. We identify a web author i with the set of web pages Ω_i he produces. Since we have no knowledge of the set of web authors, we consider every possible partion of Ω into one of more equivalence classes, $\Omega = \Omega_1 \bigcup \cdots \bigcup \Omega_a$, $\Omega_i \bigcap \Omega_j = \emptyset$ $(1 \le i \ne j \le a \le |\Omega|)$, as defining a realizable set of web authors $\mathcal{A} = 1, \ldots, a$.

11

Consider a partition of Ω into $\Omega_1, \ldots, \Omega_a$. A search term x usable by Google defines an event $\mathbf{x}_i \subseteq \Omega_i$ of web pages produced by web author i that contain search term x. Similarly, $\mathbf{x}_i \cap \mathbf{y}_i$ is the set of web pages produced by i that is returned by Google searching for pages containing both search term x and search term y. Let

$$N_i = \sum_{\{x,y\} \subseteq S} |\mathbf{x}_i \cap \mathbf{y}_i|.$$

Note that there is an $\alpha_i \geq 1$ such that $M_i \leq N_i \leq \alpha_i M_i$. For every search term $x \in S$ define a probability mass function g_i, the *individual web author's Google distribution*, on the sample space $\{\{x,y\} : x,y \in S\}$ by

$$g_i(x) = |\mathbf{x}_i|/N_i \tag{8}$$
$$g_i(x,y) = |\mathbf{x}_i \cap \mathbf{y}_i|/N_i.$$

Then, $\sum_{x \in S} g_i(x) + \sum_{x,y \in S, x<y} g_i(x,y) = 1$, where we assume a total order $\mathbf{i}$ on S to prevent summing the same term twice.

THEOREM 1 *Let $\Omega_1, \ldots, \Omega_a$ be any partition of Ω into subsets (web authors), and let $g_1, \ldots, g_a$ be the corresponding individual Google distributions. Then the Google distribution g is universal for the enumeration $g, g_1, \ldots, g_a$.*

PROOF. We can express the overall Google distribution in terms of the individual web author's distributions:

$$g(x) = \sum_{i \in \mathcal{A}} \frac{N_i}{N} g_i(x)$$
$$g(x,y) = \sum_{i \in \mathcal{A}} \frac{N_i}{N} g_i(x,y).$$

Consequently, $g(x) \geq (N_i/N) g_i(x)$ and $g(x,y) \geq (N_i/N) g_i(x,y)$. □

REMARK 6 The definition of universality above requires pointwise domination by the universal probability mass function of every enumerated probability mass function, up to a positive multiplicative constant that is independent of the properties of the graphs of the actual probability mass functions in the enumeration. The universality of a given probability mass function in the enmeration is invariant under changing the probability mass functions in the enumeration, within the obvious constraints, apart from the given universal probability mass function. Thus, for the Google distribution to be univeral for the list of individual Google distributions, it does not matter how we change the individual Google distributions, or even add new distributions, as long as the weighted sum with weights N_i/N sums up to the overal (in this case also the universal) Google distribution. This requirement is in line with the extensive body of research in universal algorithmic probability, where the list of enumerated probability mass functions is countably infinite—related to the standard enumeration of Turing machines, [19]. It intends and ensures precisely the same nontrivial properties in the case of a finite list of probability mass functions, of precisely the same quality and intension: The universality notion has the same purpose and effect for finite and infinite enumerations.

For the sake of completeness, let us make a detailed comparison with the related notions in algorithmic probability. There, one is interested in establishing a universal probability mass function for a countably infinite list of all, and only, probability mass functions thta are computable in a certain approximation sense. For details see the above reference. Here, it is relevant that all, and only, these probability mass functions can be effectively enumerated by starting from the standard effective enumeration of the list of all Turing machines, effectively modifying each Turing machine in the list so that it computes a probability mass function of the prescribed type. In this setting, it suffices to consider a series of weights $c_i > 0$ such that $\sum c_i \leq 1$, for example, $c_i = d/i^2$ with $\sum 1/i^2 = d < \infty$. For every converging infinite series $\sum c_i < \infty$ of positive terms c_i, we have that (i) the series sums to a particular positive value c, depending only on the summed series , and not on the probability mass functions being weighted; and (ii) $\lim_{i \to \infty} c_i = 0$. In the current setting of finitely many individual Google distributions, we have replaced the notion of "all countably infinitely many probability mass functions of a certain type" by "all combinations of probability mass functions possible by partitioning of Ω in all possible ways." For a finite series $\sum_{i=1}^a c_i$, item (ii) is not applicable. Item (i) is translated into $\sum_{i=1}^a c_i \geq 1$; but $\sum_{i=1}^a c_i$ can be set to any positive constant that doesn't depend on Ω and S. Then, from a certain cardinality of Ω and S onwards, the choice will be satisfactory. Here we have chosen $\sum_{i=1}^a c_i$ so as to suffice also for small cardinalities. ◇

12

REMARK 7 Let us show that, for example, the uniform distribution $L(x) = 1/s$ ($s = |\mathcal{S}|$) over the search terms $x \in \mathcal{S}$ is not universal, for $s > 2$. By the requirement $\sum c_i \geq 1$, the sum taken over the number a of web authors in the list $\mathcal{A}$, there is an i such that $c_i \geq 1/a$. Taking the uniform distribution on say s search terms assigns probability $1/s$ to each of them. Now choosing a partition of Ω in $a < s$ web authors, there is an individual Google distribution g_i with $c_i \geq 1/a$, and we must satisfy $g(x) \geq c_i g_i(x)$. By the definition of universality of a probability mass function for the list of individual Google probability mass functions g_i, we can choose the function g_i freely (as long as $a \geq 2$, and there is another function g_j to exchange probabilities of search terms with). So choose some search term x and set $g_i(x) = 1$, and $g_i(y) = 0$ for all search terms $y \neq x$. Then, we obtain $g(x) = 1/s \geq c_i g_i(x) = 1/a$. This yields the required contradiction for $s > a \geq 2$. $\Diamond$

3.4 Universality of Normalized Google Distance

Every individual web author produces both an individual Google distribution g_i, and an *individual Shannon-Fano code* G_i associated with g_i, Section 2.3, for the search terms.

DEFINITION 2 The associated *individual normalized Google distance* NGD_i of web author i is defined according to (7), with G_i substituted for G.

The normalized Google distance is *universal* for the family of individual normalized Google distances, in the sense that it is at least as small as the least individual normalized Google distance, with high probability. In Theorem 2 we show that, for every $k \geq 1$, the inequality

$$\mathrm{NGD}(x,y) < \beta\mathrm{NGD}_i(x,y) + \gamma, \tag{9}$$

with $\beta = (\min\{G_i(x), G_i(y) - \log k\}/\min\{G_i(x), G_i(y)\}$ and $\gamma = (\log kN/N_i)/\min\{G(x), G(y)\}$, is satisfied with probability going to 1 with growing k.

REMARK 8 To interpret (9), we observe that in case $G_i(x)$ and $G_i(y)$ are large with respect to $\log k$, then $\beta \approx 1$. If moreover $\log N/N_i$ is large with respect to $\log k$, then $\gamma \approx (\log N/N_i)/\min\{G(x), G(y)\}$. Let us estimate γ under reasonable assumptions. Without loss of generality assume $G(x) \leq G(y)$. If $f(x) = |\mathbf{x}|$, the number of pages returned on query x, then $G(x) = \log(N/f(x))$. Thus, $\gamma \approx (\log N - \log N_i)/(\log N - \log f(x))$. The uniform expectation of N_i is $N/|\mathcal{A}|$, and N divided by that expectation of N_i equals $|\mathcal{A}|$, the number of web authors producing web pages. The uniform expectation of $f(x)$ is $N/|\mathcal{S}|$, and N divided by that expectation of $f(x)$ equals $|\mathcal{S}|$, the number of Google search terms we use. Thus, the more the number of search terms exceeds the number of web authors, the more γ goes to 0 in expectation. $\Diamond$

REMARK 9 To understand (9), we may consider the code-lengths involved as the Google database changes over time. It is reasonable to expect that both the total number of pages as well as the total number of search terms in the Google database will continue to grow for some time. In this period, the sum total probability mass will be carved up into increasingly smaller pieces for more and more search terms. The maximum singleton and doubleton codelengths within the Google database will grow without bound. But the universality property of the Google distribution implies that the Google distribution's code length for almost all particular search terms will be within a negligible error of the best codelength among any of the individual web authors. The size of this gap will grow more slowly than the code-length for any particular search term over time. Thus, the coding space that is suboptimal in the Google distribution's code is an ever-smaller piece (in terms of proportion) of the total coding space. $\Diamond$

THEOREM 2 *(i) For every pair of search terms x, y, the set of web authors $\mathcal{B}$ for which (9) holds, has $\sum\{N_i/N : i \in \mathcal{B}\} > (1 - 1/k)^2$. That is, if we select a web page uniformly at random from the total set Ω of web pages, then we have probability $> (1 - 1/k)^2$ that (9) holds for web author i that authored the selected web page.*

(ii) For every web author $i \in \mathcal{A}$, the g_i-probability concentrated on the pairs of search terms for which (9) holds is at least $(1 - 2/k)^2$.

PROOF. The Shannon-Fano codes G_i associated with g_i satisfy $G(x) \le G_i(x) + \log N/N_i$ and $G(x,y) \le G_i(x,y) + \log N/N_i$. Therefore, substituting $G(x,y)$ by $G_i(x,y) + \log N/N_i$ in (7), we obtain

$$\text{NGD}(x,y) \le \frac{G_i(x,y) - \max\{G(x),G(y)\} + \log N/N_i}{\min\{G(x),G(y)\}}. \tag{10}$$

Markov's Inequality says the following: Let p be any probability mass function; let f be any nonnegative function with p-expected value $\mathbf{E} = \sum_i p(i)f(i) < \infty$. For $\mathbf{E} > 0$ we have $\sum_i \{p(i) : f(i)/\mathbf{E} > k\} < 1/k$.

(i) Define a probability mass function $p(i) = N_i/N$ for $i \in \mathcal{A}$. For every $x \in \mathcal{S}$, the Google probability mass value $g(x)$ equals the expectation of the individual Google probability mass values at x, that is, $\sum_{i \in \mathcal{A}} p(i)g_i(x)$. We can now argue as follows: for every x as above, we can consider x fixed and use "$g_i(x)$" for "$f(i)$", and "g(x)" for "$\mathbf{E}$" in Markov's Inequality. Then, $\sum_i \{p(i) : g_i(x)/g(x) > k\} < 1/k$, and therefore $\sum_i \{p(i) : g_i(x)/g(x) \ge k\} > 1 - 1/k$. Hence, there is a x-dependent subset of web authors $\mathcal{B}_x = \{i \in \mathcal{A} : g_i(x)/g(x) \ge k\}$, such that $\sum \{p(i) : i \in \mathcal{B}\} > 1 - 1/k$ and by definition $g_i(x) \ge kg(x)$ for $i \in \mathcal{B}_x$. Then, for $i \in \mathcal{B}_x$, we have $G_i(x) - \log k \le G(x)$. Substitute $G_i(x) - \log k$ for $G(x)$, with probability $> (1 - 1/k)$ that $G_i(x) - \log k \le G(x)$ for web author i that authored a web page that is selected uniformly at roandom from the set Ω of all web pages. Recall that $\sum \{p(i) : i \in \mathcal{B}_x\} = \sum \{N_i : i \in \mathcal{B}_x\}/N$. Similarly this holds for search term y with respect to the equivalently defined $\mathcal{B}_y$. The combination of search terms x and y therefore satisfy both $G_i(x) - \log k \le G(x)$ and $G_i(y) - \log k \le G(y)$, for $i \in \mathcal{B}$ with $\mathcal{B} = \mathcal{B}_x \cap \mathcal{B}_y$. Then, $\sum \{p_i : i \in \mathcal{B}\} = \sum \{N_i : i \in \mathcal{B}\}/N \ge \sum \{N_i : i \in \mathcal{B}_x\}/N \times \sum \{N_i : i \in \mathcal{B}_y\}/N > (1 - 1/k)^2$. Hence, the probability that both search terms x and y satisfy $G_i(x) - \log k \le G(x)$ and $G_i(y) - \log k \le G(y)$, respectively, for web author i that authored a web page selected uniformly at random from Ω, is greater than $(1 - 1/k)^2$. Substitute both $G(x)$ and $G(y)$ according to these probabilistically satisfied inequalities in (10), both in the max-term in the numerator, and in the min-term in the denominator. This proves item (i).

(ii) Fix web author $i \in \mathcal{A}$. We consider the conditional probability mass functions $g'(x) = g(x|x \in \mathcal{S})$ and $g'_i(x) = g_i(x|x \in \mathcal{S})$ over single search terms: The g'_i-expected value of $g'(x)/g'_i(x)$ is

$$\sum_x g'_i(x) \frac{g'(x)}{g'_i(x)} \le 1.$$

Then, by Markov's Inequality

$$\sum_x \{g'_i(x) : g'(x) \le jg'_i(x)\} > 1 - \frac{1}{j}. \tag{11}$$

Let $\sum_{x \in \mathcal{S}} g(x) = h$ and $\sum_{x \in \mathcal{S}} g_i(x) = h_i$. Since the probability of an event of a doubleton set of search terms is not greater than that of an event based on either of the constituent search terms, $1 \ge h, h_i \ge 1/2$. Therefore, $2g(x) \ge g'(x) \ge g(x)$ and $2g_i(x) \ge g'_i(x) \ge g_i(x)$. Then, for the search terms x satisfying (11), we have

$$\sum_x \{g_i(x) : g(x) \le 2jg_i(x)\} > 1 - \frac{1}{j}.$$

For the x's with $g(x) \le 2jg_i(x)$ we have $G_i(x) \le G(x) + \log 2j$. Substitute $G_i(x) - \log 2j$ for $G(x)$ (there is g_i-probability $\ge 1 - 1/j$ that $G_i(x) - \log 2j \le G(x)$) and $G_i(y) - \log 2j \le G(y)$ in (10), both in the max-term in the numerator, and in the min-term in the denominator. Noting that the two g_i-probabilities $(1 - 1/j)$ are independent, the total g_i-probability that both substitutions are justified is at least $(1 - 1/j)^2$. Substituting $k = 2j$ proves item (ii).

$\square$

Therefore, the Google normalized distance minorizes every normalized compression distance based on a particular user's generated probabilities of search terms, with high probability up to an error term that in typical cases is ignorable.

4 Introduction to Experiments

4.1 Google Frequencies and Meaning

In our first experiment, we seek to verify that Google page counts capture something more than meaningless noise. For simplicity, we do not use NGD here, but instead look at just the Google probabilities of small integers in several
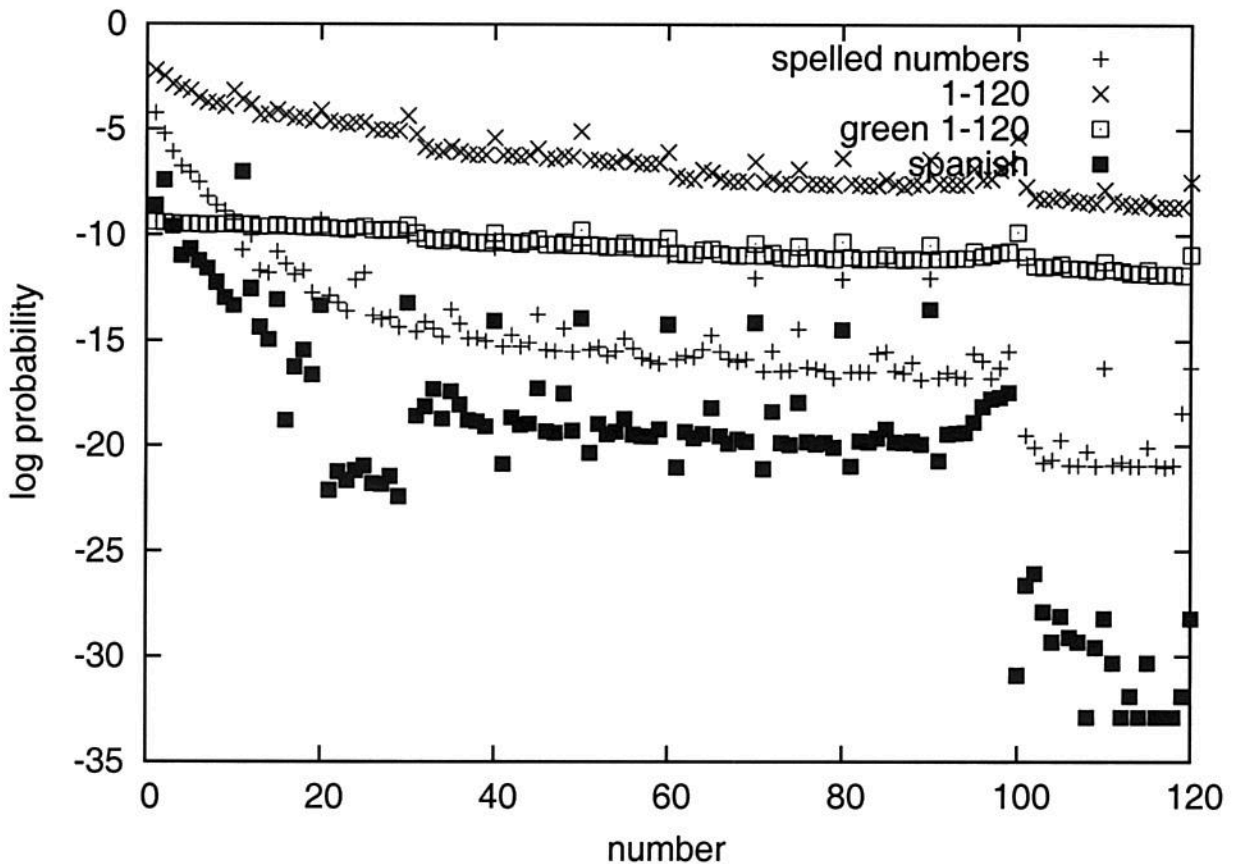
Figure 1: Numbers versus log probability (pagecount / M) in a variety of languages and formats.

formats. The first format we use is just the standard numeric representation using digits, for example "43". The next format we use is the number spelled out in English, as in "forty three". Then we use the number spelled in Spanish, as in "cuarenta y tres". Finally, we use the number as digits again, but now paired with the fixed and arbitrary search term *green* . In each of these examples, we compute the probability of search term x as $f(x)/M$. We plotted $\log(f(x)/M)$ against x in Figure 1 for x runs from 1 to 120. Notice that numbers such as even multiples of ten and five stand out in every representation in the sense that they have much higher frequency of occurrence. We can treat only low integers this way: integers of the order 10^{23} mostly don't occur since there are not web pages enough to represent a noticeable fraction of them (but Avogadro's number 6.022×10^{23} occurs with high frequency both in letters and digits).

Visual inspection of the plot gives clear evidence that there is a positive correlation between every pair of formats. We can therefore assume that that there is some underlying structure that is independent of the language chosen, and indeed the same structure appears even in the restricted case of just those webpages that contain the search term *green*
.

4.2 Some Implementation Details

Before explaining our primary NGD results, a few implementation details should be clarified. When entering searches in Google, a rich syntax is available whereby searches may be precisely constrained, see [7]. We use two important features. If you enter the term every generation in Google, it counts precisely the number of pages that contain

both the word *every* and the word *generation*, but not necessarily consecutively like every generation. If you instead enter "every generation", then this tells Google that both words must appear consecutively. Another feature that is important is the + modifier. Google ignores common words and characters such as "where" and "how", as well as certain single digits and single letters. Prepending a + before a searchterm indicates that every result must include the following term, even if it is a term otherwise ignored by Google. Experiments show that every generation and +"every" +"generation" give slightly different results, say 17,800,000 against 17,900,000. Some other experiments show, that whatever the Google manual says, the form horse rider is slightly sensitive to adding spaces, while +"horse" +"rider" is not. Therefore, we only use the latter form. Our translation from a tuple of search terms into a Google search query proceeds in three steps: First we put double-quotes around every search term in the tuple. Next, we prepend a + before every term. Finally, we join together each of the resultant strings with a single space. For example, when using the search terms "horse" and "rider", it is converted to the Google search query +"horse" +"rider".

Another detail concerns avoiding taking the logarithm of 0. Although our theory conveniently allows for ∞ in this case, our implementation makes a simplifying compromise. When returning $f(x)$ for a given search, we have two cases. If the number of pages returned is non-zero, we return twice this amount. If the pages is equal to 0, we do not return 0, but instead return 1. Thus, even though a page does not exist in the Google index, we credit it half the probability of the smallest pages that do exist in Google. This greatly simplifies our implementation and seems not to result in much distortion in the cases we have investigated.

4.3 Three Applications of the Google Method

In this paper we give three applications of the Google method: unsupervised learning in the form of hierarchical clustering, supervised learning using Support Vector Machines, and matching using correlation. For the hierarchical clustering method we refer to [5], and the correlation method is well known. For the supervised learning, several techniques are available. For the SVM method used in this paper, we refer to the excellent exposition [2], and give a brief summary in Appendix A.

5 Hierarchical Clustering

For these examples, we used our software tool available from http://complearn.sourceforge.net/, the same tool that has been used in our earlier papers [5, 4] to construct trees representing hierarchical clusters of objects in an unsupervised way. However, now we use the normalized Google distance (NGD) instead of the normalized compression distance (NCD). Recapitulating, the method works by first calculating a distance matrix using NGD among all pairs of terms in the input list. Then it calculates a best-matching unrooted ternary tree using a novel quartet-method style heuristic based on randomized hill-climbing using a new fitness objective function optimizing the summed costs of all quartet topologies embedded in candidate trees.

5.1 Colors and Numbers

In the first example, the objects to be clustered are search terms consisting of the names of colors, numbers, and some tricky words. The program automatically organized the colors towards one side of the tree and the numbers towards the other, Figure 2. It arranges the terms which have as only meaning a color or a number, and nothing else, on the farthest reach of the color side and the number side, respectively. It puts the more general terms black and white, and zero, one, and two, towards the center, thus indicating their more ambiguous interpretation. Also, things which were not exactly colors or numbers are also put towards the center, like the word "small". We may consider this an example of automatic ontology creation.

5.2 Dutch 17th Century Painters

In the example of Figure 3, the names of fifteen paintings by Steen, Rembrandt, and Bol were entered. The names of the associated painters were not included in the input, however they were added to the tree display afterward to
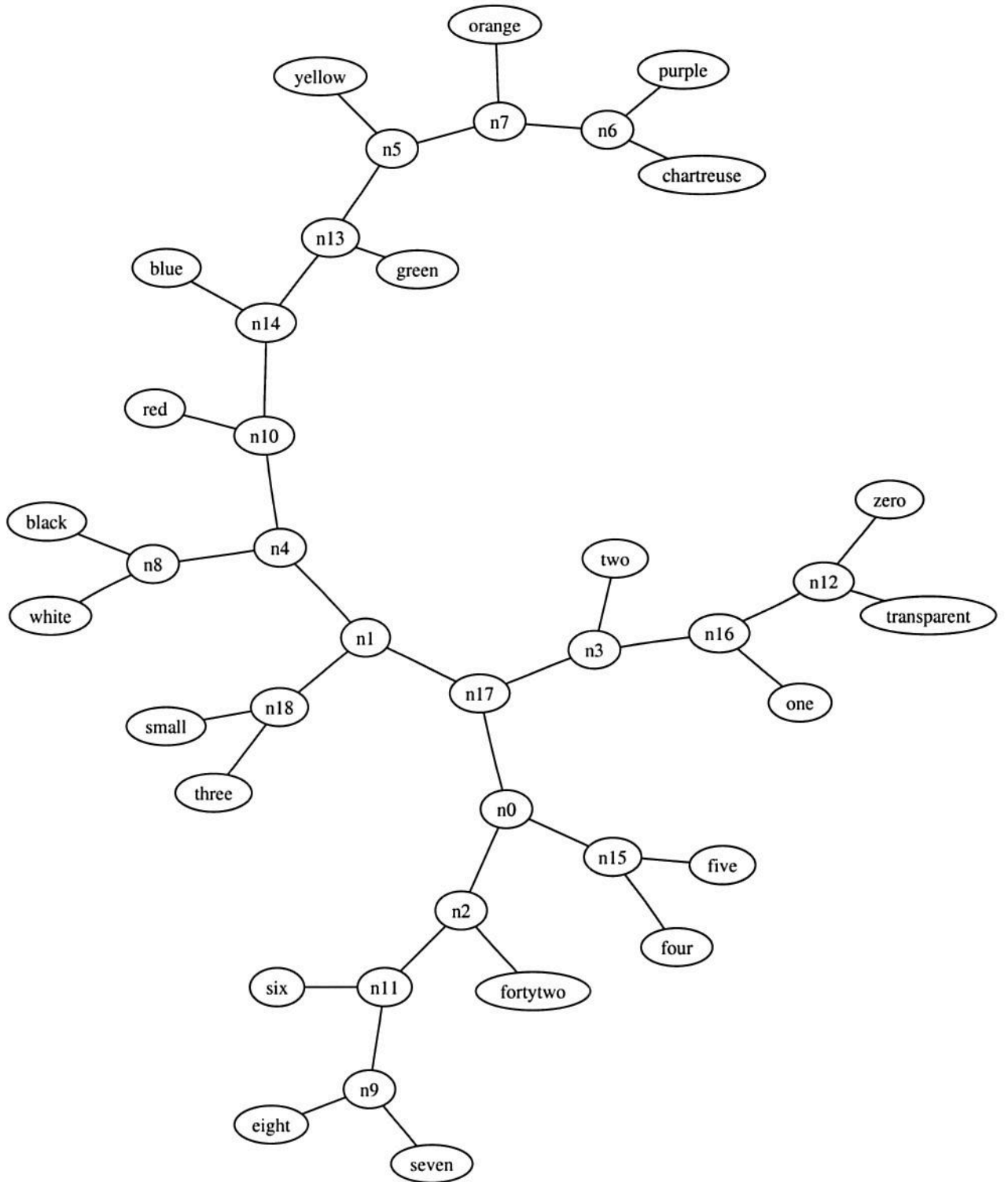
16

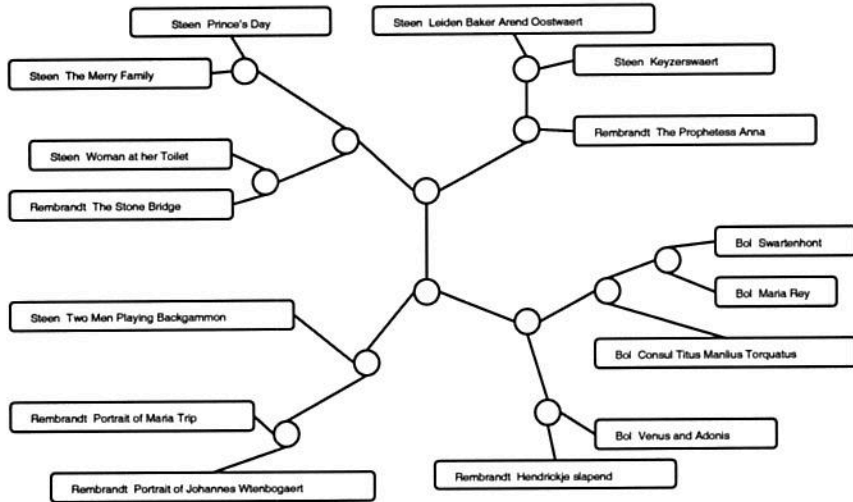Figure 2: Colors and numbers arranged into a tree using NGD .

Figure 3: Fifteen paintings tree by three different painters arranged into a tree hierarchical clustering. In the experiment, only painting title names were used; the painter prefix shown in the diagram above was added afterwards as annotation to assist in interpretation. The painters and paintings used follow. **Rembrandt van Rijn** : *Hendrickje slapend; Portrait of Maria Trip; Portrait of Johannes Wtenbogaert ; The Stone Bridge ; The Prophetess Anna* ; **Jan Steen** : *Leiden Baker Arend Oostwaert ; Keyzerswaert ; Two Men Playing Backgammon ; Woman at her Toilet ; Prince's Day ; The Merry Family* ; **Ferdinand Bol** : *Maria Rey ; Consul Titus Manlius Torquatus ; Swartenhont ; Venus and Adonis*

demonstrate the separation according to painters. This type of problem has attracted a great deal of attention [12]. A more classical solution is offered in[13], where a domain-specific database is used for similar ends. The present automatic oblivious method obtains results that compare favorably with the latter feature-driven method.

6 SVM Learning

We augment the Google method by adding a trainable component of the learning system. Here we use the Support Vector Machine (SVM) as a trainable component. For a brief introduction to SVM 's see Appendix A. We use LIBSVM software for all of our SVM experiments.

The setting is a binary classification problem on examples represented by search terms. We require a human expert to provide a list of at least 40 *training words*, consisting of at least 20 positive examples and 20 negative examples, to illustrate the contemplated concept class. The expert also provides, say, six *anchor words* $a_1, \ldots, a_6$, of which half are in some way related to the concept under consideration. Then, we use the anchor words to convert each of the 40 training words $w_1, \ldots, w_{40}$ to 6-dimensional *training vectors* $\bar{v}_1, \ldots, \bar{v}_{40}$. The entry $v_{j,i}$ of $\bar{v}_j = (v_{j,1}, \ldots, v_{j,6})$ is defined as $v_{j,i} = \mathrm{NGD}(w_i, a_j)$ ($1 \le i \le 40$, $1 \le j \le 6$). The training vectors are then used to train an SVM to learn the concept, and then test words may be classified using the same anchors and trained SVM model.

6.1 Emergencies

In the next example, Figure 4, we trained using a list of emergencies as positive examples, and a list of "almost emergencies" as negative examples. The figure is self-explanatory. The accuracy on the test set is 75%.

6.2 Learning Prime Numbers

In Figure 5 the method learns to distinguish prime numbers from non-prime numbers by example:

18

Training Data

Positive Training	(22 cases)			
avalanche	bomb threat	broken leg	burglary	car collision
death threat	fire	flood	gas leak	heart attack
hurricane	landslide	murder	overdose	pneumonia
rape	roof collapse	sinking ship	stroke	tornado
train wreck	trapped miners			

Negative Training	(25 cases)			
arthritis	broken dishwasher	broken toe	cat in tree	contempt of court
dandruff	delayed train	dizziness	drunkenness	enumeration
flat tire	frog	headache	leaky faucet	littering
missing dog	paper cut	practical joke	rain	roof leak
sore throat	sunset	truancy	vagrancy	vulgarity

Anchors	(6 dimensions)			
crime	happy	help	safe	urgent
wash				

Testing Results

	Positive tests	Negative tests
Positive Predictions	assault, coma, electrocution, heat stroke, homicide, looting, meningitis, robbery, suicide	menopause, prank call, pregnancy, traffic jam
Negative Predictions	sprained ankle	acne, annoying sister, campfire, desk, mayday, meal
Accuracy	15/20 = 75.00%	

Figure 4: Google- SVM learning of "emergencies."

Training Data

Positive Training	(21 cases)			
11	13	17	19	2
23	29	3	31	37
41	43	47	5	53
59	61	67	7	71
73				

Negative Training	(22 cases)			
10	12	14	15	16
18	20	21	22	24
25	26	27	28	30
32	33	34	4	6
8	9			

Anchors	(5 dimensions)			
composite	number	orange	prime	record

Testing Results

	Positive tests	Negative tests
Positive Predictions	101, 103, 107, 109, 79, 83, 89, 91, 97	110
Negative Predictions		36, 38, 40, 42, 44, 45, 46, 48, 49

Accuracy   18/19 = 94.74%

Figure 5: Google- SVM learning of primes.

The prime numbers example illustrates several common features of our method that distinguish it from the strictly deductive techniques. It is common for our classifications to be good but imperfect, and this is due to the unpredictability and uncontrolled nature of the Google distribution.

6.3   WordNet Semantics: Specific Examples

To create the next example, we used WordNet. WordNet is a semantic concordance of English. It also attempts to focus on the meaning of words instead of the word itself. The category we want to learn, the concept, is termed "electrical", and represents anything that may pertain to electronics, Figure 6. The negative examples are constituted by simply everything else. This category represents a typical expansion of a node in the WordNet hierarchy. The accuracy on the test set is 100%: It turns out that "electrical terms" are unambiguous and easy to learn and classify by our method.

In the next example, Figure 7, the concept to be learned is "religious". Here the positive examples are terms that are commonly considered as pertaining to religious items or notions, the negative examples are everything else. The accuracy on the test set is 88.89%. Religion turns out to be less unequivocal and unambiguous than "electricity" for our method.

Notice that what we may consider to be errors, can be explained, or point at, a secondary meaning or intention of these words. For instance, some may consider the word "shepherd" to be full of religious connotation. And there has been more than one religion that claims to involve "earth" as a component. Such examples suggest to use the method for exploratory semantics: establishing less common, idiosyncratic, or jargon meaning of words.

6.4   WordNet Semantics: Statistics

The previous examples show only a few hand-crafted special cases. To investigate the more general statistics, a method was devised to estimate how well the NGD -Google- SVM approach agrees with WordNet in a large number of automatically selected semantic categories. Each automatically generated category followed the following sequence.

First we must review the structure of WordNet; the following is paraphrased from the official WordNet documentation available online. WordNet is called a semantic concordance of the English language. It seeks to classify words into many categories and interrelate the meanings of those words. WordNet contains synsets. A synset is a synonym set; a set of words that are interchangeable in some context, because they share a commonly-agreed upon meaning with little or no variation. Each word in English may have many different senses in which it may be interpreted; each of these distinct senses points to a different synset. Every word in WordNet has a pointer to at least one synset. Each synset, in turn, must point to at least one word. Thus, we have a many-to-many mapping between English words and synsets at the lowest level of WordNet. It is useful to think of synsets as nodes in a graph. At the next level we have lexical and semantic pointers. Lexical pointers are not investigated in this paper; only the following semantic pointer types are used in our comparison: A semantic pointer is simply a directed edge in the graph whose nodes are synsets. The pointer has one end we call a *source* and the other end we call a *destination*. The following relations are used:

1. *hyponym* : X is a hyponym of Y if X is a (kind of) Y.

2. *part meronym* : X is a part meronym of Y if X is a part of Y.

3. *member meronym* : X is a member meronym of Y if X is a member of Y.

4. *attribute* : A noun synset for which adjectives express values. The noun *weight* is an attribute, for which the adjectives *light* and *heavy* express values.

5. *similar to* : A synset is similar to another one if the two synsets have meanings that are substantially similar to each other.

Using these semantic pointers we may extract simple categories for testing. First, a random semantic pointer (or edge) of one of the types above is chosen from the WordNet database. Next, the source synset node of this pointer is used as a sort of root. Finally, we traverse outward in a breadth-first order starting at this node and following only edges that have an identical semantic pointer type; that is, if the original semantic pointer was a hyponym, then we would only follow hyponym pointers in constructing the category. Thus, if we were to pick a hyponym link initially

Training Data

Positive Training (58 cases)

Cottrell precipitator	Van de Graaff generator	Wimshurst machine	aerial	antenna
attenuator	ballast	battery	bimetallic strip	board
brush	capacitance	capacitor	circuit	condenser
control board	control panel	distributer	electric battery	electric cell
electric circuit	electrical circuit	electrical condenser	electrical device	electrical distributor
electrical fuse	electrical relay	electrograph	electrostatic generator	electrostatic machine
filter	flasher	fuse	inductance	inductor
instrument panel	jack	light ballast	load	plug
precipitator	reactor	rectifier	relay	resistance
security	security measures	security system	solar array	solar battery
solar panel	spark arrester	spark plug	sparking plug	suppresser
transmitting aerial	transponder	zapper		

Negative Training (55 cases)

Andes	Burnett	Diana	DuPonts	Friesland
Gibbs	Hickman	Icarus	Lorraine	Madeira
Quakeress	Southernwood	Waltham	Washington	adventures
affecting	aggrieving	attractiveness	bearer	boll
capitals	concluding	constantly	conviction	damming
deeper	definitions	dimension	discounting	distinctness
exclamation	faking	helplessness	humidly	hurling
introduces	kappa	maims	marine	moderately
monster	parenthesis	pinches	predication	prospect
repudiate	retry	royalty	shopkeepers	soap
sob	swifter	teared	thrashes	tuples

Anchors (6 dimensions)

bumbled	distributor	premeditation	resistor	suppressor
swimmers				

Testing Results

	Positive tests	Negative tests
Positive Predictions	cell, male plug, panel, transducer, transformer	
Negative Predictions		Boswellizes, appointer, enforceable, greatness, planet
Accuracy	10/10 = 100.00%	

Figure 6: Google- SVM learning of "electrical" terms.

Training Data

Positive Training	(22 cases)			
Allah	Catholic	Christian	Dalai Lama	God
Jerry Falwell	Jesus	John the Baptist	Mother Theresa	Muhammad
Saint Jude	The Pope	Zeus	bible	church
crucifix	devout	holy	prayer	rabbi
religion	sacred			

Negative Training	(23 cases)			
Abraham Lincoln	Ben Franklin	Bill Clinton	Einstein	George Washington
Jimmy Carter	John Kennedy	Michael Moore	atheist	dictionary
encyclopedia	evolution	helmet	internet	materialistic
minus	money	mouse	science	secular
seven	telephone	walking		

Anchors	(6 dimensions)			
evil	follower	history	rational	scripture
spirit				

Testing Results

	Positive tests	Negative tests
Positive Predictions	altar, blessing, communion, heaven, sacrament, testament, vatican	earth, shepherd
Negative Predictions	angel	Aristotle, Bertrand Russell, Greenspan, John, Newton, Nietzsche, Plato, Socrates, air, bicycle, car, fire, five, man, monitor, water, whistle
Accuracy	24/27 = 88.89%	

Figure 7: Google- SVM learning of "religious" terms.

that says a *tiger* is a *cat*, we may then continue to follow further hyponym relationships in order to continue to get more specific types of cats. See the WordNet homepage [23] documentation for specific definitions of these technical terms. For examples of each of these categories consult the experiments listed in the Appendix at [3].

Once a category is determined, it is expanded in a breadth first way until at least 38 synsets are within the category. If the category cannot be expanded this far, then a new one is chosen. Once a suitable category is found, and a set of at least 38 members has been formed, a training set is created using 25 of these cases, randomly chosen. Next, three are chosen randomly as anchors. And finally the remaining ten are saved as positive test cases. To fill in the negative training cases, random words are chosen from the WordNet database. Next, three random words are chosen as unrelated anchors. Finally, 10 random words are chosen as negative test cases.

For each case, the SVM is trained on the training samples, converted to 6-dimensional vectors using NGD . The SVM is trained on a total of 50 samples. The kernel-width and error-cost parameters are automatically determined using five-fold cross validation. Finally testing is performed using 20 examples in a balanced ensemble to yield a final accuracy.

There are several caveats with this analysis. It is necessarily rough, because the problem domain is difficult to define. There is no protection against certain randomly chosen negative words being accidentally members of the category in question, either explicitly in the greater-depth transitive closure of the category, or perhaps implicitly in common usage but not indicated in WordNet. In several cases, such as "radio wave" and "DC" in the "big science" experiment, there appears to be an arguable case to support the computer's classification in cases where this phenomenon occurs. Another detail to notice is that WordNet is available through some web pages, and so undoubtedly contributes something to Google pagecounts. Further experiments comparing the results when filtering out WordNet images on the web suggest that this problem doesn't usually affect the results obtained, except when one of the anchor terms happens to be very rare and thus receives a non-negligible contribution towards its page count from WordNet views. In general, our previous NCD based methods, as in [5], exhibit large-granularity artifacts at the low end of the scale; for small strings we see course jumps in the distribution of NCD for different inputs which makes differentiation difficult. With the Google-based NGD we see similar problems when page counts are less than a hundred.

We ran 100 experiments. The actual data are available at [3]. A histogram of agreement accuracies is shown in Figure 8. On average, our method turns out to agree well with the WordNet semantic concordance made by human experts. The mean of the accuracies of agreements is 0.8725. The variance is ≈ 0.01367, which gives a standard deviation of ≈ 0.1169. Thus, it is rare to find agreement less than 75%. These results confirm that we are able to perform a rudimentary form of *generalization* within a *conceptual domain* programmatically using Google. For hand-crafted examples it performed comparably, and so this suggests that there may be latent semantic knowledge. Is there a way to use it?

7   Matching the Meaning

Yet another potential application of the NGD method is in natural language translation. (In the experiment below we don't use SVM 's to obtain our result, but determine correlations instead.) Suppose we are given a system that tries to infer a translation-vocabulary among English and Spanish. Assume that the system has already determined that there are five words that appear in two different matched sentences, but the permutation associating the English and Spanish words is, as yet, undetermined. This setting can arise in real situations, because English and Spanish have different rules for word-ordering. Thus, at the outset we assume a pre-existing vocabulary of eight English words with their matched Spanish translation. Can we infer the correct permutation mapping the unknown words using the pre-existing vocabulary as a basis? We start by forming an NGD matrix using additional English words of which the translation is known, Figure 9. We label the columns by the translation-known English words, the rows by the translation-unknown words. The entries of the matrix are the NGD 's of the English words labeling the columns and rows. This constitutes the English basis matrix. Next, consider the known Spanish words corresponding to the known English words. Form a new matrix with the known Spanish words labeling the columns in the same order as the known English words. Label the rows of the new matrix by choosing one of the many possible permutations of the unknown Spanish words. For each permutation, form the NGD matrix for the Spanish words, and compute the pairwise correlation of this sequence of values to each of the values in the given English word basis matrix. Choose the permutation with the highest positive correlation. If there is no positive correlation report a failure to extend the
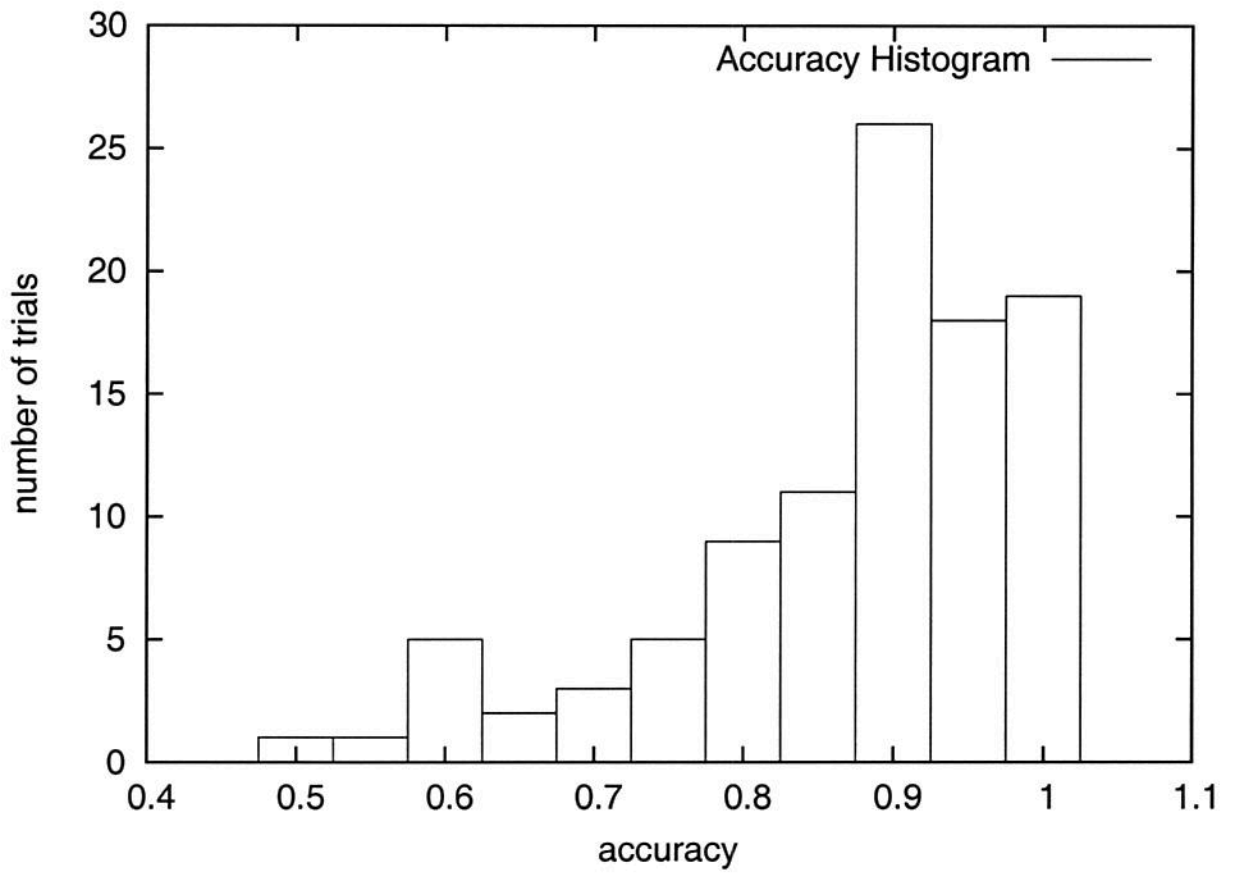
Figure 8: Histogram of accuracies over 100 trials of WordNet experiment.

Given starting vocabulary

English	Spanish
tooth	diente
joy	alegria
tree	arbol
electricity	electricidad
table	tabla
money	dinero
sound	sonido
music	musica

Unknown-permutation vocabulary

plant	bailar
car	hablar
dance	amigo
speak	coche
friend	planta

Figure 9: English-Spanish Translation Problem

	English	Spanish
Predicted (optimal) permutation	plant	planta
	car	coche
	dance	bailar
	speak	hablar
	friend	amigo

Figure 10: Translation Using NGD

vocabulary. In this example, the computer inferred the correct permutation for the testing words, see Figure 10.

8 Conclusion

A comparison can be made with the *Cyc* project [14]. Cyc, a project of the commercial venture Cycorp, tries to create artificial common sense. Cyc's knowledge base consists of hundreds of microtheories and hundreds of thousands of terms, as well as over a million hand-crafted assertions written in a formal language called CycL [20]. CycL is an enhanced variety of first-order predicate logic. This knowledge base was created over the course of decades by paid human experts. It is therefore of extremely high quality. Google, on the other hand, is almost completely unstructured, and offers only a primitive query capability that is not nearly flexible enough to represent formal deduction. But what it lacks in expressiveness Google makes up for in size; Google has already indexed more than eight billion pages and shows no signs of slowing down.

Epistemology: In the case of context-free statistical compression such as gzip , we are trying to approximate the Kolmogorov complexity of a string. Another way of describing the calculation is to view it as determining a probability mass function (viewing the compressed string as Shannon-Fano code, Section 2.3), approximating the *universal distribution*, that is, the negative exponential of the Kolmogorov complexity [19]. The universal probability of a given string can equivalently be defined as the probability that the reference universal Turing machine outputs the string if its input program is generated by fair coin flips. In a similar manner, we can associate a particular Shannon-Fano code, the *Google code*, with the Google probability mass function. Coding every search term by its Google code, we define a "Google compressor." Then, in the spirit of Section 3.2, we can view the Google probability mass function as a universal distribution for the individual Google probability mass functions generated by the individual web authors, substituting "web authors" for "Turing machines".

Concerning the SVM method: The Google-SVM method does not use an individual word in isolation, but instead uses an ordered list of its NGD relationships with fixed anchors. This then removes the possibility of attaching to the isolated (context-free) interpretation of a literal term. That is to say, the inputs to our SVM are not directly search terms, but instead an image of the search term through the lens of the Google distribution, and relative to other fixed terms which serve as a grounding for the term. In most schools of ontological thought, and indeed in the WordNet database, there is imagined a two-level structure that characterizes language: a many-to-many relationship between word-forms or utterances and their many possible meanings. Each link in this association will be represented in the Google distribution with strength proportional to how common that usage is found on the web. The NGD then amplifies and separates the many contributions towards the aggregate page count sum, thereby revealing some components of the latent semantic web. In almost every informal theory of cognition we have the idea of connectedness of different concepts in a network, and this is precisely the structure that our experiments attempt to explore.

Universality: The Google distribution is a comparable notion, in the context of the world-wide-web background information, to the universal distribution: The universal distribution multiplicatively dominates all other distributions in that it assigns a higher weight to some elements when appropriately scaled. This suggests that it covers everything without bound. Google surely represents the largest publicly-available single corpus of aggregate statistical and indexing information so far created. Only now has it been cheap enough to collect this vast quantity of data, and it

26

seems that even rudimentary analysis of this distribution yields a variety of intriguing possibilities. One of the simplest avenues for further exploration must be to increase training sample size, because it is well-known that SVM accuracy increases with training sample size. It is likely that this approach can never achieve 100% accuracy like in principle deductive logic can, because the Google distribution mirrors humankind's own imperfect and varied nature. But it is also clear that in practical terms the NGD can offer an easy way to provide results that are good enough for many applications, and which would be far too much work if not impossible to program in a foolproof deductive way.

The Road Ahead: We have demonstrated that NGD can be used to extract meaning in a variety of ways from the statistics inherent to the Google database. So far, all of our techniques look only at the page count portion of the Google result sets and achieve surprising results. How much more amazing might it be when the actual contents of search results are analyzed? Consider the possibility of using WordNet familiarity counts to filter returned search results to select only the least familiar words, and then using these in turn as further inputs to NGD to create automatic discourse or concept diagrams with arbitrary extension. Or perhaps this combination can be used to expand existing ontologies that are only seeded by humans. Let us list some of the future directions and potential application areas:

1. There seems to also be an opportunity to apply these techniques to generic language acquisition, word sense disambiguation, knowledge representation, content-filtration and collaborative filtering, chat bots, and discourse generation.

2. There are potential applications of this technique to semi-intelligent user-interface design; for predictive completion on small devices, speech recognition, or handwriting recognition.

3. A user interface possibility is the idea of concept-class programming for non-programmers, or software to form a conceptual predicate by way of example without forcing the user to learn a formal programming language. This might be used, for example, in a network content filtration system that is installed by non-programmer parents to protect their young children from some parts of the internet. Or perhaps an IT manager is able to adjust the rule determining if a particular email message is a well-known virus and should be filtered without writing explicit rules but just showing some examples.

4. How many people are able to write down a list of prime numbers as shown in an earlier test case, Figure 5, compared to how many people are able to write a program in a real programming language that can calculate prime numbers? Concept clustering by example is significantly simpler than any formal programming language and often yields remarkably accurate results without any effort at hand-tuning parameters.

5. The colors versus numbers tree example, Figure 2, is rife with possibilities. A major challenge of the Semantic Web and XML as it stands is in integrating diverse ontologies created by independent entities [9]. XML makes the promise of allowing seamless integration of web services via customized structured tags. This promise is for the most part unrealized at this point on the web, however, because there is not yet sufficient agreement on what sets of XML tags to use in order to present information; when two different parties each build databases of recipes, but one organizes the recipes according to their country of origin and another according to their sweetness or savory flavor, these two databases cannot "understand" one another insofar as they may exchange recipes. XML allows us to format our data in a structured way, but fails to provide for a way for different structure conventions to interoperate. There have been many attempts to solve this and none have been satisfactory. Usually solutions involve mapping the separate schemas into some sort of global schema and then creating a global standardization problem that requires significant coordinated effort to solve. Another approach is to create a meta-language like DAML that allows for automatic translation among certain very similar types of ontologies, however this requires a great deal of effort and forethought on the part of the schema designers in advance and is brittle in the face of changing ontologies. By using NGD we may create a democratic and natural ontology for almost any application in an unsupervised way. Furthermore, if instead we want finer control over the ontological organization, then a human expert may define a custom ontology and then NGD may be used to provide a normal, global, and automatic reference frame within which this ontology may be understood without additional costly human effort. So, for example, NGD may be used in the recipe example above, Figure 9, 10, to automatically "understand" the difference between a Chinese or Mediterranean recipe, and could thus be used to automatically translate between the two conflicting ontologies.

27

6. Another future direction is to apply multiple concurrent binary classifiers for the same classification problem but using different anchors. The separate classifications would have to be combined using a voting scheme, boosting scheme, or other protocol in an effort to boost accuracy.

Acknowledgments

A Appendix: Support Vector Machines

Support Vector Machines have gained enormous popularity in machine learning in recent years [2]. They represent a way to learn a classification or regression problem by example, and are comparable to neural networks in that they have the capacity to learn any function. They are large-margin classifiers. They take as input a list of k-dimensional vectors, and output a single scalar value. In order to learn, an SVM solves a convex optimization problem that is closely related to a simpler classification engine termed the separating hyperplane. In this setting, we are given a set of k-dimensional vectors x_i each labeled y_i which is 1 or -1 according to the classification. For a particular problem, a discriminating hyperplane w is one that creates a decision function that satisfies the constraint for all i:

$$y_i(x_i \cdot w + b) - 1 \geq 0$$

If no such separating hyperplane exists, then we term the learning problem *linearly inseparable* . Many real-world learning problems, such as the famous exclusive-or function, are linearly inseparable. There are many strategies for dealing with this issue. Support Vector Machines use a nonlinear kernel function to address this issue. By creating a kernel function, $k(x,y)$ that satisfies the *Mercer condition*, we may substantially enhance the power of the separating hyperplane. A kernel function defines an inner-product on the input space, and then this inner product may be used to calculate many higher-power terms of combinations of samples in the input space. This forms a higher-dimensional space, and it is well-known that once this space is made large enough, there will be a separating hyperplane. In our SVM experiments, we use a Radial Basis Function (RBF) kernel. This allows the SVM to learn any function given enough training data.

There are two parameters that control the learning of the SVM . The first relates to the kernel function. An RBF, or Radial Basis Function, kernel assumes the value 1 whenever the two input vectors are equal. If they are unequal, it decays slowly towards 0 in a radially symmetric way:

$$K(x_i, x_j) = e^{-\|x_i - x_j\|^2 / 2\alpha^2}$$

Here, α is a parameter that controls the rate of decay or width of the kernel function. Because of the exponential form, the effective dimension of an RBF kernel is potentially infinite and thus this kernel can be used to approximate any continuous function to an arbitrary degree of accuracy. This parameter must be set before the learning can begin with an SVM . Another parameter relates to how misclassified points are handled in the training data; Though it is always possible to simply continue to make the kernel width smaller and the expanded space larger until the SVM becomes essentially a lookup-table, this is often not the best strategy for learning. An alternative is to define a cost parameter and allow this to adjust the tolerance for misclassified points in the training data. This allows the SVM to generalize well even in the presence of noisy data. This cost parameter, often called c, must also be defined before training can begin.

We select α and c using a grid searching technique. For each of these parameters, it is appropriate to search dozens of powers of two. Together, this creates a grid with hundred of different parameter settings. We use five-fold cross-validation to select which of these grid points defines the optimal parameter setting. First the data is divided into

five random partitions: A, B, C, D, E. Then, for each candidate parameter setting or grid point, we run five different training runs. On the first run, we train on B, C, D, and E, and then we determine an accuracy using part A. Next, we train on A, C, D, E and test with B. We continue in this way and then average all five test scores to arrive at an estimate of how well the learning process performed. Once this process is done for all training data, we may just choose one of the grid points that attains a maximum accuracy.

References

[1] C.H. Bennett, P. Gács, M. Li, P.M.B. Vitányi, W. Zurek, Information Distance, *IEEE Trans. Information Theory*, 44:4(1998), 1407–1423.

[2] C.J.C. Burges. A tutorial on support vector machines for pattern recognition, *Data Mining and Knowledge Discovery*, 2:2(1998),121–167.

[3] Automatic Meaning Discovery Using Google: 100 Experiments in Learning WordNet Categories, 2004, http://www.cwi.nl/~cilibrar/googlepaper/appendix.pdf

[4] R. Cilibrasi, R. de Wolf, P. Vitanyi. Algorithmic clustering of music, *Computer Music Journal*, 2004.

[5] R. Cilibrasi, P. Vitanyi. Clustering by compression, Submitted to *IEEE Trans. Information Theory*. http://www.archiv.org/abs/cs.CV/0312044

[6] J.-P. Delahaye, Classer musiques, langues, images, textes et genomes, *Pour La Science*, 317(March 2004), 98–103.

[7] The basics of Google search, http://www.google.com/help/basics.html.

[8] L.G. Kraft, A device for quantizing, grouping and coding amplitude modulated pulses. Master's thesis, Dept. of Electrical Engineering, M.I.T., Cambridge, Mass., 1949.

[9] L. Lakshmanan, F. Sadri. Xml interoperability, *Proc. Intn'l Workshop Web and Databases (WebDB)* San Diego, California, June 2003.

[10] H. Muir, Software to unzip identity of unknown composers, *New Scientist*, 12 April 2003.

[11] K. Patch, Software sorts tunes, *Technology Research News*, April 23/30, 2003.

[12] L. Rutledge, M. Alberink, R. Brussee, S. Pokraev, W. van Dieten, M. Veenstra. Finding the Story — Broader Applicability of Semantics and Discourse for Hypermedia Generation, *Proc. 14th ACM Conf. Hypertext and Hypermedia* Nottingham, UK, pp. 67-76, August 23-27, 2003

[13] M.J. Alberink, L.W. Rutledge, M.J.A. Veenstra, Clustering semantics for hypermedia presentation, *CWI Tech Report*, INS-E0409, ISSN 1386-3681, 2004.

[14] D. B. Lenat. Cyc: A large-scale investment in knowledge infrastructure, *Comm. ACM*, 38:11(1995),33–38.

[15] A.N. Kolmogorov. Three approaches to the quantitative definition of information, *Problems Inform. Transmission*, 1:1(1965), 1–7.

[16] A.N. Kolmogorov. Combinatorial foundations of information theory and the calculus of probabilities, *Russian Math. Surveys*, 38:4(1983), 29–40.

[17] M. Li, J.H. Badger, X. Chen, S. Kwong, P. Kearney, and H. Zhang, An information-based sequence distance and its application to whole mitochondrial genome phylogeny, *Bioinformatics*, 17:2(2001), 149–154.

[18] M. Li, X. Chen, X. Li, B. Ma, P. Vitanyi. The similarity metric, *IEEE Trans. Information Theory*, 50:12(2004), 3250- 3264.

[19] M. Li, P. M. B. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*, 2nd Ed., Springer-Verlag, New York, 1997.

[20] S. L. Reed, D. B. Lenat. Mapping ontologies into cyc. *Proc. AAAI Conference 2002 Workshop on Ontologies for the Semantic Web*, Edmonton, Canada. http://citeseer.nj.nec.com/509238.html

[21] D.H. Rumsfeld, The digital revolution, originally published June 9, 2001, following a European trip. In: H. Seely, The Poetry of D.H. Rumsfeld, 2003, http://slate.msn.com/id/2081042/

[22] C. E. Shannon. A mathematical theory of communication. *Bell Systems Technical J.*, 27(1948), 379–423 and 623–656.

[23] G.A. Miller et.al, WordNet, A Lexical Database for the English Language, Cognitive Science Lab, Princeton University, http://www.cogsci.princeton.edu/ wn