Challenges in the Information Sciences for Statisticians

By Siddhartha R. Dalal

Telcordia Technologies

Information Sciences and Technologies areas are the fastest growing areas affecting almost every facet of human life. The statistics profession has played a visible role in areas like agricultural, demographical and biological sciences. However, the role of Statistics and the Statistics profession is not as visible in Information Sciences and Technologies at large. This paper argues that, to the extent that Statisticians are willing and able to expand their role beyond the standard modeling, sampling and designing roles, they will be enormously successful in this new field. However, if they do not take a larger view and think about unconventional problems and innovative solutions, they are likely to be left behind by experts trained in other Information Sciences related disciplines. As examples this paper describes two problems in Information Sciences and Technologies related to search engine technology and software testing technology. On the surface, traditional statistical theories do not have much to offer for solving them and Statisticians looking for conventional solutions would have not been successful in solving these problems. However, unconventional statistical thinking played a critical role in solving these problems.

1. Foreword It is a pleasure to be a part of this conference in honor of Professor W. J. Hall at the occasion of his 70th birthday. Jack Hall and I moved to University of Rochester in the same year. He came there as the chairman of the newly established Statistics Department and I came as a student. I was most impressed by Jack's interest in many different areas of Statistics and his willingness to supervise my thesis in an area outside of his primary research interest. He had a sharp eye for what was practical and doable. When I left Rochester, I asked him for a final word of advice. He mentioned that he was quite impressed by my curiosity about everything, and that continuing being curious will stand me well in my research life. This has become one of my maxims.

I embarked on my industrial research career at Bell Laboratories in 1980. Since then I have spent most of my career at Telcordia Technologies, a leading telecommunications research and development company formerly known as Bellcore. Much of my time is spent on Information Sciences. One of the great things about working in industry is that challenging problems abound. I have written this paper to describe of some of these "challenging" problems and suggest changes in paradigm for statisticians to be successful in the new Information Science industry.

2. Introduction To underscore the point of "challenging" problems, I would like to briefly describe a small sub-sample, two "challenging" problems that my colleagues and I have worked on at Telcordia Technologies. They are in the areas of

- Software testing, and

- Search engine and text mining techniques for Information Retrieval.

I hope Jack would feel good about the way that the field of statistics continues to play a major but atypical role in Information Science today. Though both of these problems are unconventional as far as traditional theories of Statistics are concerned, successfully solving them required statistical thinking. In fact, solutions of both of these have started new research areas in their respective fields.

With the explosion of Information, many new unconventional approaches are needed to solve problems. To the extent that Statisticians are willing and able to expand their role beyond the standard modeling, sampling and designing roles, they will be enormously successful. However, if they do not take a larger view and think about unconventional problems and innovative solutions, they are likely to be left behind by experts trained in other Information Sciences related disciplines.

3. Software Testing Software testing is a very expensive part of building sophisticated software systems such as those used in modern telecommunications operations. A major reason for emphasis on testing is that the systems must meet very high reliability and availability standards. The actual cost may be one third or more of the total cost of building the software. Naturally, questions arise as to how much testing is really necessary, and how the testing can be done in the most efficient manner. For a statistical formulation of how much testing is necessary I refer to Dalal and Mallows [8] and Dalal and McIntosh [10]. I now concentrate on how the testing can be done in the most efficient manner.

Since the number of possible inputs is typically very large, testers need to select a sample, commonly called a suite, of test cases, based on effectiveness and adequacy. To test software, combinations of all inputs must be provided and the output from the software be checked against the corresponding correct output. Each combination tested is called a test case. One would like to generate test cases that include inputs over a broad range of permissible values.

Much testing is done in an intuitive and less formal manner. Typically, testers, working from their knowledge of the system under test and of the prospective users, decide on a set of specific inputs. Clearly, there is the possibility that important interactions among the inputs will be missed. Herein lie significant opportunities for a systematic approach, based on ideas from sampling and experimental design theory. Consider the following example.

Example 1. Testing an Air to Ground Missile System: Consider a software system controlling the state of an air to ground missile. The key inputs for the software are the altitude, attack and bank angles, speed, pitch, roll, and yaw. (There are many more- e.g. ambient temperature, pressure, wind velocity, etc., I will consider these later on.) Typically, these variables do not have any joint constraints as far as the software is concerned.

To determine the test cases, we can go back to the requirement document. Suppose we are interested in testing the response during attack maneuvering. We select the boundary values, maximum and minimum, as the two representative values for each of the seven input variables and denote them symbolically as 1 and 2, respectively. Then in the language of statistical experimental design, we have seven

factors, A,..., G (altitude, attack angle, bank angle, speed, pitch, roll, and yaw), each at two levels. To test all the possible combinations, one would need a complete factorial experiment, which would have $2^7 = 128$ test cases consisting of all possible sequences of 1's and 2's.

In this example, it may be possible to test all 128 test cases, but this may be only one of many systems that need to be tested. Further, if we want to increase the scope of the testing by including three more variables (for example H: ambient temperature, I: pressure, and J: wind velocity) then we would have $2^{10} = 1024$ test cases. In anything other than toy problems, the situation is typically much worse than this, and even using automated testing, the number of cases is impossibly large. I will illustrate this by giving two more examples.

Example 2. Screen Testing: Typically, users of business systems interact with software via a succession of screens, each of which has a number of fields. It is not uncommon to have 50 or more fields; for example Cohen, Dalal, Kajla, and Patton [4] give an example of a screen with 76 fields. Assuming only 2 values per field, (for example "valid" and "missing"), one has 2^{76} test cases. At a rate of a million cases per second (impossible to achieve today even with automation), this would require 2×10^{15} years to test.

Example 3. Interoperability Testing: Periodically, software companies update their products, and sell them as new versions (e.g. Windows 95 vs. Windows 3.1). When these products come out, it is essential that they work with a number of versions of other software products. Thus, the issue of interoperability testing is critical. For example, one would want Windows NT and XP, Word 2000 and 1998, Excel 5.0 and 2000, etc, all to work with one another. Thus, suppose one had four software products, and wanted to support two versions of each, then we must study 16 interoperability problems. However each problem represents a large number of subproblems, and detailed analysis would result in a huge number of factors.

There are many other examples showing the geometric explosion in the number of test cases- see Dalal and Patton [7] for feature interactions testing and Burroughs, Jain and Erickson [2], for protocol testing. Sloane [11] references applications on hardware testing, including circuit and network testing.

Empirical data and experience suggest that most faults are associated with an error in a single field or a combination of a pair of fields. Faults attributable to more complex combinations are atypical and do not involve more than 3 or 4 fields. Thus, higher order combinations can be sacrificed to gain efficiency. This type of thinking is similar to that of statistical experimental design where one tends to focus on main effects and low-order interactions. In a statistical experiment if one is interested in only main effects, then one can use a highly fractionated factorial design to reduce number of runs. However, in the case of software testing, there is no interest in estimating additive effects. Interest lies in covering the test space as completely as possible and checking whether the test cases pass or fail. However, it is certainly possible to use standard statistical designs. For example consider the set of test cases given in Table 1. This is an orthogonal array of strength two, with 7 factors and 2 levels. It requires 8 test cases instead of 128. It is clear that all possible pairwise combinations of levels of two factors are covered in a balanced way (exactly twice each in this example). Thus testing according to this design will protect against

Table 1: Test Cases for Example 3 using Orthogonal Array for 7 fields at 2 levels

Factor	A	B	C	D	E	F	G
Test 1	1	1	1	1	1	1	1
Test 2	1	1	1	2	2	2	2
Test 3	1	2	2	1	1	2	2
Test 4	1	2	2	2	2	1	1
Test 5	2	1	2	1	2	1	2
Test 6	2	1	2	2	1	2	1
Test 7	2	2	1	1	2	2	1
Test 8	2	2	1	2	1	1	2

Table 2: Test Cases by Combinatorial Design Method for 10 fields with 2 levels

Factor	A	B	C	D	E	F	G	H	I	J
Test 1	1	1	1	1	1	1	1	1	1	1
Test 2	1	1	1	1	2	2	2	2	2	2
Test 3	1	2	2	2	1	1	1	2	2	2
Test 4	2	1	2	2	1	2	2	1	1	2
Test 5	2	2	1	2	2	1	2	1	2	1
Test 6	2	2	2	1	2	2	1	2	1	1

any incorrect implementation of the code involving any pairwise interaction, and also whatever other higher order interactions happen to be represented in the table. Brownlie, Prowse and Phadke [1] have suggested the use of orthogonal array designs of strength two for testing.

However, in software testing repeating a run with exactly the same inputs will give exactly the same output, so exact replication is unhelpful, and wasteful. The problem is not that of estimating an unknown response function, but rather that of determining whether the software functions correctly under all relevant input conditions; the response is either "O.K.", in which case nothing needs to be done, or "failure" in which case the test case can be analyzed to determine the cause (or causes) of the failure. The efficiency of a test design is measured by the degree to which it covers the relevant input space. If we insist on coverage of all pairs, but give up the restriction of balance, we can do a lot better. For example, Table 2 gives a design that achieves pairwise coverage of 10 two-level factors, in just 6 test cases. These designs were first proposed by Dalal and Patton [7] based on a system they first proposed called AETG System. For further references on AETG System, refer to papers by Cohen *et al.* [4, 5, 3, 6] where they referred the new paradigm as Combinatorial Designs. Dalal and Mallows [8] in a review paper called this type of designs as Factor Covering Designs and proposed several new construction methods.

Thus, in the missile example, besides incorporating the 7 factors already mentioned, one can now include three more factors and still guarantee protection against any pairwise interaction, using only 6 test cases.Unlike orthogonal arrays, for which the number of cases grows at least linearly with the number of factors, covering de-

signs grow at only a logarithmic rate. Further, orthogonal designs typically require equal number of levels, do not always exist, and cannot be easily constructed to give constraints. The savings are more dramatic when the number of factors is large; for example with 126 binary factors we can cover all pairwise interactions with only 10 runs, whereas an orthogonal array would require at least 128 runs. The exhaustive testing would require 2^{126} test cases. It is interesting to observe that while the problem itself is not statistical at all: there is no statistical data, no uncertainty, no statistical model, no statistical inference; statistical thinking is critical for the solution. Further, this is a nice example of statistical thinking leading to a highly visible contribution to the commercial world. Not only is the methodology in use in Telcordia, a CMM level 5 software supplier, but it is also available on web as an automatic test generation service. More information on this service is available at `http://aetgweb.argreenhouse.com`, including more complex examples and other references.

4. Latent Semantic Indexing (LSI) Since computers are replacing humans in performing tedious tasks, they need to be equipped with the ability to process natural language. A Telcordia patented technology called Latent Semantic Indexing (LSI) was invented for Natural Language Processing (NLP) when the web was just in its infancy. This technology underlies several search engines. Deerwester *et al.* [9] provides a detailed discussion.

Basically, an English document can be represented by a term frequency vector; elements of the vector represent counts of corresponding words that appear in the document. Since there are about 50,000 typical words in English, a collection of 1,000 documents can be represented by a term by document matrix (TDM) as large as 50,000 by 1,000. We will refer to a column of TDM as term frequency vector frequency of a term which appears in the corresponding document, and a row of TDM as document frequency vector how many times a document in the collection contains the corresponding term.

One of the commonly used methods to search for relevant documents is called keyword search. It consists of retrieving all the documents that contain any of the words in a query. This is equivalent to thinking about the query as a document in TDM and then retrieving all the documents which have positive dot product (i.e., cosine of the angle) with the query. One can score the documents according to the magnitude of the normalized inner product. Although keyword searches can be effective, they are not perfect. In particular, choosing the right word from a host of synonyms can mean the difference between success and failure. For example, transparencies, viewgraphs, foils, overheads, slides, etc., are referring the same item, though, particular document may only use one of these terms.

Specifically, a) keyword searches bring up huge number of documents which may not be relevant because same words can have different meaning in different context (precision), and more importantly, b) it also misses a lot of document which do not have the exact terms in query, but are synonymous (recall). Several studies have shown that key search methods have around 50% precision and 25% recall. The difficulty is that from statistical perspective textual data is hard to model and word matching is not very effective.

Latent Semantic Indexing (LSI, in short) basically operates in the same manner as keyword search- it uses TDM and does score in the same manner. LSI, however, makes some modifications to the TDM that yield an ability to anticipate synonyms automatically. This also reduces the dimensions of TDM.

Specifically, to be able to efficiently process and extract global information in the documents, LSI gets rid of inconsistencies and ambiguities that individual documents have shown and only maintains the most important underlying structure of the original TDM. It takes the full TDM, performs a Singular Value Decomposition (SVD), and selects k most influential singular vectors to give a lower rank approximation to the original TDM. More specifically, let be a t by d TDM (t terms and d documents). Then SVD will decompose into the product of three matrices,

$$X_{td} = T_{tm} S_{mm} D'_{md},$$

where $T(D)$ has orthogonal unit length column vectors referred to as the left (right) singular vectors and S is a diagonal matrix of positive singular values in decreasing order. Here m is the rank of X_{td}. The first k largest singular values are retained while others are set to 0. Consequently, when the three matrices are multiplied back to obtain an approximation to X_{td} only the corresponding k left and right singular vectors need to be retained. Therefore the rank of X_{td} is effectively reduced from m to k. The choice of k is rather subjective. How large the k is depends on how close we want the approximation to be and how different in magnitude the singular values are to each other. Evidently, we choose k to be equal to m if we want an exact "approximation". On the other hand, k can be a small number if the sum of the first few leading singular values is relatively large compared to the sum of all the singular values, e.g. 80%. In practice, the latter case seldom occurs, so a general guideline is to set k to be 300 for medium-sized documents. Let $\hat{X}_{td}$ denote the lower rank approximation to X_{td}. Then

$$\hat{X}_{td} = \hat{T}_{tk} \hat{S}_{kk} \hat{D}'_{kd}.$$

But what is a reasonable numeric representation for a document? What is a reasonable numeric representation for a term? Answers to these questions are similar. I will try to answer the first one, and the answer to the second one will follow right through with similar arguments. Suppose two documents are similar, then the pattern of their term frequency vector will be similar. If we take the inner product of their term frequency vectors, we obtain a larger value than if they were dissimilar. Therefore, element i, j element of $X'_{dt} X_{dt}$ can serve as a measure of the similarity between ith and jth documents. But

$$X'_{dt} X_{dt} = (D_{dm} S_{mm} T'_{mt})(T_{tm} S_{mm} D'_{md}) = (D_{dm} S_{mm})(S_{mm} D'_{md}),$$

so we can treat rows of $D_{dm} S_{mm}$ as a numeric representation for the documents when we want to compare between documents. After dimension reduction, rows of $\hat{D}_{dk} \hat{S}_{kk}$ are approximate numeric representations of the documents. A new document or query can be projected into the LSI document vector space by $\hat{d}^* = d^* \hat{T}_{tm} \hat{S}^{-1}_{mm}$, where d^* is a row vector of term frequencies of the new document or query. This defines a simple version of the LSI document indexing process.

Similarly, rows of $T_{tm} S_{mm}$ can serve as the numeric representation of terms when we want to compare between terms. Therefore, rows of $\hat{T}_{tk} \hat{S}_{kk}$ are approximation

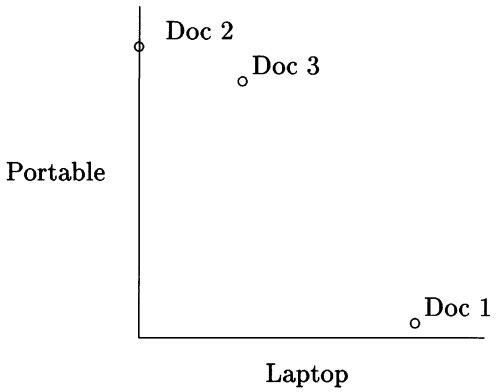

Figure 1: Keyword Search: 3 documents containing terms Portable and Laptop

to the numerical representation for the terms and a new term can be projected into the LSI term vector space by $\hat{t}^* = t^* \hat{D}_{dm} \hat{S}_{mm}^{-1}$, where t^* is a row vector of document frequencies for the new term.

Since we have numerical representations for all documents (terms), we can measure similarity between documents (terms) by some similarity measure, such as cosine of the angle between two LSI document (term) vectors. This is same as in the keyword search method.

4.1 An illustration of LSI

LSI has been used in many applications and technologies similar to it have become underpinning of several new search engines. Consider an example, which consists of three documents. Doc 1 and Doc 3 focuses on displays used on portable computers. However, Document 1 refers to only Laptop and Document 3 refers to Portable computer. Document 2 talks in general about technology and peripherally refers to Laptop and Portable in the same vein. If one were now to have a query consisting of the term Laptop, as depicted in Figure 1, one will retrieve Doc 1 and Doc 2, but Doc 3 would be completely left out. However, if one were to use LSI with 2 dimensions, given the cross reference supplied by Doc 2 between Laptop and Portable, we would get higher scores for Doc 1 and Doc 3 compared to Doc 2. Since we have $k = 2$, we can see this more clearly. We can plot rows of $D_{d2}S_{22}$ to examine relationship between documents. Figure 2a plots this relationship and shows that Doc 1, Doc 3 and Query are all close to each other compared to Doc 2. Similarly, in Figure 2b we plot rows of $D_{dm}S_{mm}$ to examine the relationships between words. Laptop and Portable are clearly closer to each other than Display, etc. For other more detailed examples, we refer to Deerwester *et al.* [9] and a book by Berry and Browne (1999). Additional references and a demo is available on the web at `http://lsi.argreenhouse.com/lsi/`

While the model itself is not treated as a stochastic model and there is no statistical inference, statistical thinking along the line of principal components and factor analysis is at the heart of the solution. This is yet another nice example of statistical thinking leading to a major contribution to the commercial world. Several search engines (e.g. Excite) are using variations of this approach.

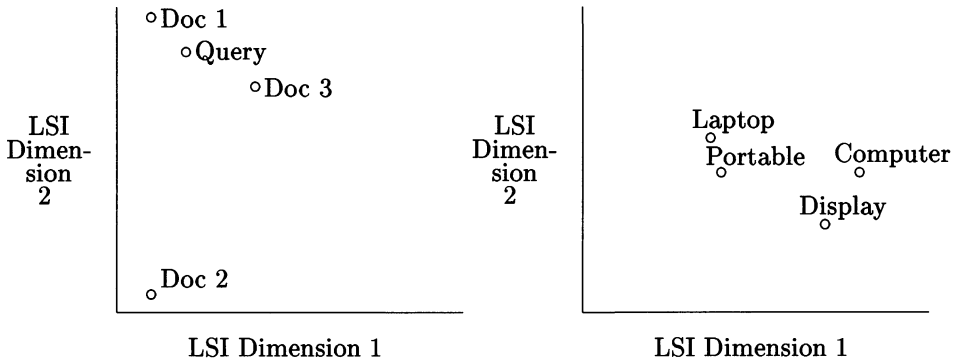

Figure 2a and 2b: LSI retrieval: 2a. Similar documents retrieved. 2b. Similar words retrieved

REFERENCES

[1] R. Brownlie, J. Prowse, and M. Phadke. Robust testing of AT&T PMX/StarMail using OATS. *AT&T Technical Journal*, 71(3):41–47, 1992.

[2] K. Burroughs, A. Jain, and R. L. Erickson. Improved quality of protocol testing through techniques of experimental design. In *Supercomm/ICC '94 (IEEE International Conf. on Communications)*, pages 745–752, 1994.

[3] D. M. Cohen, S. R. Dalal, M. Fredman, , and G. C. Patton. The AETG system: An approach to testing based on combinatorial design. *IEEE Transactions of Software Engineering*, 23, 1997.

[4] D. M. Cohen, S. R. Dalal, A. Kajla, and G. C. Patton. The automatic efficient test generator (AETG) system. In *Proceedings of the 5th International Symposium on Software Reliability Engineering*, Los Alamitos, CA, 1994. IEEE Computer Society Press.

[5] D. M. Cohen, S. R. Dalal, J. Parelius, and G. C. Patton. The combinatorial approach to automatic test generation. *IEEE Software*, 13(5):83–89, 1996.

[6] D. M. Cohen and Fredman M. New techniques for designing qualitatively independent systems. Technical Report DCS-96-114, Rutgers University, 1996.

[7] S. R. Dalal and Patton G. C. Automatic efficient test generator (AETG): A test generation system for screen testing, protocol verification and feature interactions testing. Technical report, Bellcore, 1993.

[8] S. R. Dalal and C. L. Mallows. When should one stop testing software? *Journal of the American Statistical Association*, 83:872–879, 1988.

[9] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, and R.A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41:391–407, 1990.

[10] Dalal S. R. and McIntosh A. M. When to stop testing for large software systems with changing code. *IEEE Transactions of Software Engineering*, 20:318–323, 1994.

[11] N. J. A. Sloane. Covering arrays and intersecting codes. *Journal of Combinatorial Designs*, 1:51–63, 1993.

IMAGING AND SERVICES TECHNOLOGY CENTER
XEROX CORPORATION
800 PHILLIPS ROAD
M/S 0128-53E
WEBSTER, NY 14580
sdalal@crt.xerox.com