Let’s say law enforcement needs to locate all references to “incendiary devices” in a computerized stack of documents.

Or MegaHugeCorp is planning a merger with GigaHugeCorp, and the Federal Trade Commission has asked MegaHugeCorp for all materials on file relating to “marketplace competitive analysis.”

Or let’s say that during a high-speed chase through the streets of Prague, a CIA operative is able to wrest a single typed paragraph out of the hands of a suspected member of a terrorist organization. The CIA needs to find other possible matches to this text from millions of satellite and wire intercepts.

Or let’s say the whole family is coming to dinner, and you need the blueberry pie recipe you typed into your laptop last year.

In the first scenario, the obvious text search is for *incendiary devices*, while in the CIA case a search for a few keywords from the stolen paragraph might work. In the MegaHugeCorp case, *marketplace competitive analysis* might be the search. And to locate the pie, the obvious search string is *blueberry*.

How can dtSearch®, for example, search over terabytes of text in less than a second? It does so by building an index that stores the location of each word within a document. Once an index is complete, search time is generally less than a second, even through millions of files.

And these searches would yield ... NOTHING! No “incendiary devices” in the first scenario. Nothing in the merger case: “I’m sorry Federal Trade Commission, but we’ve remarkably gotten to dominate our marketplace without any apparent thought to ‘marketplace competitive analysis.’” Nothing in the blueberry pie example: “I knew my dog was eating computer files!” And in the CIA case, 42
billion documents, which is far too many retrieved documents for even a pack of trained chimps to thumb through.

This is where advanced query techniques, available in dtSearch, for example, are helpful. In the “incendiary devices” case, concept searching and stemming provide the solution. In the merger case, Boolean and proximity searching do the trick. In the CIA case, the solution is relevancy-ranked natural language searching and variable term weighting. And in the blueberry pie case, the answer is fuzzy searching because you misspelled blueberry.

Concept Searching and Stemming

Concept searching, also known as synonym or thesaurus searching, expands a single search request into multiple conceptual dimensions. For example, with concept searching, a search for incendiary automatically expands (using the search program’s built-in thesaurus) to include such synonyms as arsonist and inflammatory. Going broader into “related words” provides combustible and bomb.

An additional option is to find out the names of specific incendiary devices and enter them as synonyms in a user-defined addendum to the built-in thesaurus. The combined built-in thesaurus and user-defined thesaurus allows automatic synonym expansion covering a wide range of search terms, all with a simple search request.

Now, thanks to concept searching, it is easy to find every document that includes any synonym of incendiary. But what about derivatives of these words, such as inflammatories in addition to inflammatory? This type of expansion requires stemming.

Stemming uses a built-in algorithm that is familiar with the native language, in this case English, to expand a search request to include word derivatives automatically. A search for apply that includes stemming finds applies, applied, applying, but not appliance.

This brings up an important point of search request formation. When not highly familiar with the target documents (which is the case in all of the examples above, with the possible exception of the blueberry pie recipe), then cast the net broadly. But not too broadly. Choosing all words related to incendiary, or even all related words of related words, retrieves documents with terms no more relevant than felon or outlaw.

The fundamental principle of text queries is: when searching a very large and diverse database, as search completeness or retrieving all the potentially relevant information increases, so do “false hits” or the retrieval of irrelevant documents. Maximizing the chance of retrieving “the smoking gun” in a search, while minimizing the retrieval of irrelevant documents, requires casting the net just right.

Boolean, Phrase, Proximity, Wildcards, Field, Numeric Range

If concept searching and stemming were the only search tools, then it would be tough casting the net just right. Boolean operatives such as and, or and not help refine the search request. For example, in the merger scenario, a search for market analysis or competitive analysis is a basic Boolean search in combination with a phrase search. Such a search retrieves all documents that contain either the phrase market analysis or the phrase competitive analysis or both.

But what if you also need to find documents containing text such as: the market that would be relevant for the analysis. Finding that type of text requires market near analysis or competitive near analysis. How near? Let’s say within 25 words, giving the resulting query: (market or competitive) w/25 analysis. This query finds all documents that contain either the term market or the term competitive relatively near to analysis.

Wildcards, which complement a Boolean search, include the question mark (?) for replacing a single letter in a
word, and the asterisk (*) for replacing any number of letters in a word. Suppose MegaHugeCorp’s previous names were MegaMediumCorp and MegaSmallCorp. A search for Mega*Corp retrieves all three. (Almost all text searches should be case insensitive. With the possible exception of source code searching, case-sensitive searches are usually a bad idea since they miss too many relevant words.)

Another element that works well with Boolean searches is numeric range searching, such as searching for any number between 11 and 127. Field searching, or limiting a search to a specific document field, also works well with Boolean searching.

Combining Boolean searches with stemming and concept searching is also powerful. For example, with stemming on, the search request (market or competitive) w/25 analysis retrieves not only market, but also markets and marketing.

**Narrow, Broaden and Exclude**

Although it is possible to arrive at a query such as (market or competitive) w/25 analysis through logical deduction alone, a dose of trial and error is often necessary. This is particularly true if a high degree of familiarity with the target database is lacking. In the merger scenario, it is unlikely that anyone has previously seen every single document relating to “marketplace competitive analysis.”

Suppose the query (market or competitive) w/25 analysis finds a slew of documents pertaining to the analysis of supermarket shoppers. The documents date to a time when MegaHugeCorp considered offering its non-food wares in supermarkets but then rejected the idea. Because neither merging company presently sells through supermarkets, these documents fall outside of the Federal Trade Commission’s document request.

Trial and error results in a narrowed search request: ((market or competitive) w/25 analysis) and not supermarket. This finds the same document set as the previous search request, excluding all documents that contain the word supermarket. The search results represent a subset of the previous search. Alternatively, a search request that creates a slightly larger subset, by excluding only documents that contain the word supermarket within 75 words of the market/competitive/analysis.
cluster is: \((\text{market or competitive}) \ w/25 \text{ analysis}) \ \text{not} \ w/75 \text{ supermarket}.

Suppose the search request \( ((\text{market or competitive}) \ w/25 \text{ analysis}) \ \text{and not} \ \text{supermarket}) \) requires expansion to include the search term \text{exclusionary}. Broadening the search request creates a superset of the previous search. Effectively, this takes the previous search request and adds an “or” element: \(((\text{market or competitive}) \ w/25 \text{ analysis}) \ \text{and not} \ \text{supermarket}) \ \text{or} \ \text{exclusionary}.

Finally, after painstaking review of every retrieved document in the search request \(((\text{market or competitive}) \ w/25 \text{ analysis}) \ \text{and not} \ \text{supermarket}) \ \text{or} \ \text{exclusionary} \) the legal department suggests adding the term \text{monopoly}. An “anything but” search such as \text{monopoly} \ \text{and not} \ (((\text{market or competitive}) \ w/25 \text{ analysis}) \ \text{and not} \ \text{supermarket}) \ \text{or} \ \text{exclusionary} \) ensures retrieving only new files excluded from the previous search.

**Natural Language Searching**

Until now, all search requests have been structured or Boolean, with keywords such as \text{market}, \text{competitive}, \text{analysis}, \text{supermarket} and \text{exclusionary}, and structural connectors such as \text{or}, \text{and}, \text{w/25} and \text{not}. Boolean is great for searches involving a clear idea of what meets the terms of a search request. But what if the CIA, for example, has only a general sense of looking for some type of document match? In that case, relevancy-ranked natural language searching, also known as query-by-example, is a possible solution. Suppose the CIA operative retrieved the following block of text representing, remarkably, a terrorist limited warranty:

> any and all other representations and warranties, express or implied, including but not limited to implied warranties of merchantability, fitness for a particular purpose, including without limitation, whether blue bird succeeds in flying over the orange house, are expressly excluded and disclaimed.

To find a document that contains the closest match to this text—perhaps a document containing draft negotiations involving this paragraph—with natural language searching requires simply cutting and pasting this entire paragraph into a search request. Natural language searching then retrieves matching documents according to their relevancy, with the document having the highest relevancy ranking first.

The natural language search, using a query in raw format like the above paragraph, singles out keywords: \text{representations}, \text{warranties}, \text{express}, \text{implied}, etc. The search ignores connectors and other “noise” words: \text{any}, \text{and}, \text{all}, \text{other}, etc. The search then finds the documents containing the closest match to the keywords, taking into account the density and the rarity of hits.

For example, if \text{express} appears in 2 million documents and \text{warranties} appears in only two, then \text{warranties} would have a much higher relevancy weighting. Natural language searching is also combinable with stemming and concept searching. These options yield \text{warranty} and \text{warranties} as well as \text{guarantee} and \text{guarantees}.

Besides its status as one of the most advanced search types, natural language searching is also the easiest. For example,
Indexed vs. Unindexed Searching

How can dtSearch, for example, search over terabytes of text in less than a second? It does so by building an index that stores the location of each word within a document. Once an index is complete, search time is generally less than a second, even through millions of files.

dtSearch also allows unindexed and combination indexed/unindexed searching. These search options are useful for a single pass through material to discover if there is any relevant information. For example, unindexed searching might be useful to a law enforcement agency that, after confiscating a stack of hard drives, wants to know if any data on them is pertinent to a criminal investigation. Although unindexed searching is much slower than indexed searching, it is faster to do a single unindexed search than to build a search index and then do an indexed search.

<table>
<thead>
<tr>
<th>dtSearch Search Type</th>
<th>Indexed</th>
<th>Unindexed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concept / Synonym / Thesaurus</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Fielded Data</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Phrase</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Boolean</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Proximity and directed proximity</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Wildcard</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Numeric range</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Macro</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Stemming (finds variations on endings, like applies, applied, applying in a search for apply)</td>
<td>Yes (fuzziness is not &quot;hardwired&quot; into the index, making it adjustable at the time of search)</td>
<td>Yes</td>
</tr>
<tr>
<td>Phonic</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Fuzziness (adjusts from 0 to 10 for fine-tuning fuzziness to the level of OCR or typographical errors in files—a search for alphabet with a fuzziness of 1 would find alphaqet; with a fuzziness of 3, it would find both alphaqet and alpkaqet)</td>
<td>Yes (fuzziness is not &quot;hardwired&quot; into the index, making it adjustable at the time of search)</td>
<td>Yes</td>
</tr>
<tr>
<td>Natural language searching, with vector-space relevancy ranking</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Variable term weighting</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Unicode</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
requests. With \((\text{market or competitive}) \text{ w/25 analysis})\) and not supermarket, an alternative to using \text{and not} supermarket to exclude all documents that contain the word supermarket would be to search for \((\text{market or competitive}) \text{ w/25 analysis})\) and supermarket:\text{-10}. This downweights supermarket without excluding it entirely.

**Fuzzy and Phonic Searching**

And now for the missing blueberry pie recipe. After all of these complex Boolean and natural language search requests, a simple search for blueberry should be a piece of well ... pie. A search for blueberry, with stemming on, finds the entry whether it is blueberry or blueberries. But suppose blueberry is mispelled “bluegerry.” Boolean searches alone could not easily come up with the correct document.

The answer is fuzzy searching. Turning on search fuzziness to a low level finds words that match one or two deviations in letters: bluegerry, blugerry, etc. Turning on fuzziness to a higher level finds words with even more deviations in letters: blubber and bluster.

Once again, there is a direct correspondence between retrieving all possible word variations and generating “false hits.” For this reason, it’s usually best to do the search first with a low level of fuzziness, and only if that doesn’t work, to increase to a higher level of fuzziness. Note that with fuzzy searching, a misspelled search term can also find a search term that is spelled correctly in the original document.

Fuzzy searching is useful for text with spelling errors, such as typographical and OCR errors. For sound-alike errors, phonic searching can also come in handy. For example, a search for Smith finds Smythe with phonic searching.

Both fuzzy and phonic searching are combinable with Boolean, natural language and other search features. Just in case the rest of the world also can’t spell, all search requests in the previous sections are combinable with fuzzy searching.

Please visit dtSearch online at www.dtsearch.com

---

A dtSearch search for blueberry with a fuzzy level of 1 would retrieve bluegerry as well as blueberry.