Approximate Matching

COMP90049 Knowledge Technologies

Justin Zobel and Rao Kotagiri (CIS)

Semester 1

THE UNIVERSITY OF MELBOURNE

Closest match

A common computational task is to find the nearest match to a given string:

- Spelling correction, name matching, query repair, phonetic matching, data cleansing, genomics.

Further challenge: find strings with the nearest match to any of their substrings. (Why is this harder?)

Methods that address this task are "knowledge technologies" as defined in this subject.

- Thorough or exhaustive solutions (in some contexts) require unrealistic computational resources, so we need to find workable approximations.
- Correctness isn't well-defined.

Another perspective: this is *search with uncertainty*.

Search with uncertainty

Consider a file of names such as

K. Many Chandy, Jayadev Misra
Susanne Graf, Joseph Sifakis
D. Harel, A. Pnueli
Koichi Furukawa, Katsumi Niita, Yuji Matsumoto
Z. Farkas, P. Szeredi, E. Santane-Toth
Agneta Eriksson, Anna-Lena Johansson, Sten-Ake Tarnlund
Luis Moniz Pereira, Antonio Porto
Lynette Hirchmann, Karl Puder
Yuji Matsumoto, Hozumi Tanaka, Masaki Kiyono
J. Darlington, A.J. Field, H. Pull
Masahiko Sato, Tatakfumi Sakurai
P.A. Subrahmanyam, Jia-Huai You
Joseph A. Goguen, Jose Meseguer
Yonathan Malachi, Zohar Manna, Richard Waldinger

How to find names in this file, if the precise spelling is unknown?

Search with uncertainty

Again, context matters: not just language and alphabet, but the reason that the search is taking place.

In designing or choosing a search method, three factors need to be considered:

- What the user is trying to achieve.
- The source of reference material for “correct” strings.
- Tractability and interactivity – is a user waiting for a response? How will the solution behave as the uncertainty increases?

Tools: ispell, agrep, web search interfaces.

Example: the `agrep` tool is an implementation of an algorithm for matching with errors.

Pattern matching tools and spelling correction

The motivating use of pattern matching tools such as `agrep` is to identify likely matching lines in a file. No assumptions are made about meaning of characters.

The aim of spelling correction is to:

- Parse text into "words". (We'll look at parsing in another lecture.)
- Ignore the words that are found in a dictionary.
- Find the best matches for the remainder.

`... proceed if you can see no ther option ...`

Matches with `agrep -1 ther`: % with one correction
`ether`, `her`, `other`, `thar`, `the`, `thee`, `their`, `them`, `then`, `there`, `therm`, `thew`, `they`, `tier`

Observe the behaviour of agrep as *k* (number of corrections: insertions, deletions or substitutions/)is increased.

Pattern matching tools and spelling correction

A straightforward process, but it does expose underlying concepts:

- *Efficiency* and *ranking*.

Ranking. "Best match" is relative. Whether a string is a match to a query is not a binary (yes/no) decision, and the goal should be to present matches according to perceived likelihood of their being correct.

Note that the task inherently requires human participation.

What other tasks use ranking?

Pattern matching tools and spelling correction

Efficiency. Exhaustive file search is $O(n)$ in time, for say a dictionary of n entries.

Most words should be found in the dictionary; there are plenty of data structures that provide effective $O(1)$ confirmation of whether a particular string is an exact match (e.g., hashing or tries).

(Yes, I do mean *tries*, not *trees*.)

Ballpark: 10^6 to 10^7 searches per second on a laptop.

$O(n)$ may be unsatisfactory for the approximate search task. Perhaps more importantly, processing a file includes overheads such as parsing text into lines – doing so for each word to be checked is a waste of resources.

Pattern matching tools aren't customized for parsing; they are not efficient at eliminating exact matches; and don't rank results. Conclusion: Despite the task similarity, they are not suitable for spell checking.

Aside: stemming

There are two kinds of dictionaries used in spelling correction.

- *Literal:* a set of strings that must be matched exactly.
- *Root+stem:* a set of root words, a set of suffixes, and a set of rules for applying suffixes to roots.

Root: `fill`, `apply`, `invoke`.

Word:
`filled`, `filling`, but not `fillication`
`applied`, `application`, but not `apple`
`invoked`, `invoking`, `invocation`, but not `invokery`

Stemming is the process of stripping suffixes automatically. It allows a dictionary to be generalized, but many root-suffix combinations are not correct spellings.

Observe that such technologies are highly language-dependent.

Justin Zobel © University of Melbourne, 2011.

Neighbourhood search

Given a query string that is not in the dictionary, the task is to find the nearest neighbours.

One approach is to generate all neighbours, and see if they are present.

- Enumerate all of the single-character deletions, insertions, and replacements.

This generates all strings whose distance from the query is 1.

For example, for an alphabet of four letters {`e h r t`}, the word `thr` has the immediate neighbours {`hr tr th ethr hthr rthr tthr tehr thhr trhr tthr ther thhr thrr thtr thre thrh thrr thrt ehr hhr rhr ter trr ttr the thh tht`}

How many neighbours? Alphabet size A, query length n:

$$N_1 = n + A \cdot (n + 1) + (A - 1) \cdot n$$

n is due to one character deletions; A.(n+1) is due to one character insertions; (A-1).n is due to one character replacements; Ballpark: $A = 26$, $n = 8$, so $N_1 = 441$ and $\gg$ 1000 unknown strings can be processed per second.

Looks feasible! This is how `ispell` works.

Justin Zobel © University of Melbourne, 2011.

Costs

The cost of neighbourhood search is $O(n)$ since the alphabet is fixed.

What happens if we want to increase the distance from the query? For example, distance of at most 2:

$$N_2 = n + n \cdot (n - 1) + A \cdot (n + 1) + A \cdot (n + 1)A \cdot (n + 2) + \cdots$$

This does not include combinations of deletion, replacement, and insertion. n is due to one character deletions; n.(n-1) is due to two character deletions; A.(n+1) is due to one character insertions; A.(n+1)A.(n+2) is due to two character insertions;

Ballpark: $N_2 \gg 50,000$, and cost is $O(n^2)$ with a high (≈ 600) constant factor.

This combinatoric analysis indicates that exhaustive search beyond the immediate (distance$=1$) neighbourhood is at the upper limit of feasibility – and usability.

Neighbourhood size

Query is `ther`. What is going on here?

`aer aether ahem anther aver bather beer bier bother char cheer chef chert chew chez cither daer deer dither doer dyer either etcher ether ever ewer father fer gather ghee goer he hear heir hem hen hep her herb herd here herl herm hern hero hers hes hew hex hey hither hoer jeer kier lather leer lither mither mother nether oer omer other others otter over oyer peer per phew pier pother rather rhea seer she shear shed sheer sherd shes shew shier shoer shyer steer suer taed tael tahr taker taler tamer taper tar tater taxer tea tear tee teed teem teen tees ten term tern tether thaler than thar that thaw the thebe theca thee theed thees theft thegn their theirs them theme then thenar theory there therm therms these theta thew thews thewy they thief thin third this tho thorn thorp thou three threw thru thud thug thus thy tie tied tier tiers ties tiger tiler timer tither toe toea toed toes toner toper tor torr toter tother tour tower tree tref trek tret trey trier truer tsar tuber tuner twee twerp tzar user usher utter veer weer wether whee when where whet whew whey whir wither yer zither`

Justin Zobel © University of Melbourne, 2011.

Combinatoric search

Current practical techniques use variants of binary search, where:

- Strings are sorted, that is, grouped by prefix.
- Strings are inspected character-by-character until it is clear that their distance from the query is greater than k.
- Due to the sorting, groups of strings that share a prefix can be considered simultaneously.

Since the string space is extremely sparse (of the possible A^n strings, only a tiny fraction are actually observed), and there are good special-purpose data structures (e.g., suffix arrays), the cost in practice is around that of neighbourhood search at distance 1.

The distance limit k must be chosen, or repeated search is required. Additional work is required for ranking.

Because there is no sorting of a dictionary that ensures that similar strings are close together, it is still not obvious what the best solution to this search problem is.

Justin Zobel ©
University of Melbourne, 2011.

Aside: Spelling correction in context

In principle, it might seem appealing to use grammar or morphological structure to help guide spelling correction.

But there are significant obstacles:

- Relatively few spelling errors can be resolved by context.
- The cost of building the dictionary would be prohibitive.

There are good contextual spelling checkers for typical errors, based on both machine learning and manual analysis of syntax and error patterns. These are most effective for short words.

There are no spelling checkers that use broad context, such as the category or topic of a document (however this might be defined). We are probably stuck with order-0 (context-free) correction for most words. But we can try and be smarter about what we try to correct . . .

Justin Zobel © University of Melbourne, 2011.

Edit distances

From another perspective: take the query string and each candidate match, and find the *edit distance* between them.

The simplest edit distance is the number of insertions, deletions, and substitutions needed to turn one string into another.

The smaller the edit distance, the closer the match.

```
Gorbach–ev
       |   |
Gorbechyov
```

```
Connell
      |
Con–al–
```

Measures such as edit distances are attractive because matches are scored, so answers can be ranked.

Edit distances

Consider the alignment of two strings `agt` and `aact` using a simple scheme where an alignment of two matching characters scores $+1$, an alignment of two mismatching characters scores -1, and an insertion or deletion of a character also scores -1.

(There is no explicit penalty for sequences being of unequal lengths).

Intuitively, using this simple scoring scheme, an optimal alignment of the two sequences would be: $\begin{bmatrix} \text{aact} \\ \text{–agt} \end{bmatrix}$

Justin Zobel © University of Melbourne, 2011.

Edit distances

Assume an array `F` of size $|q| \times |t|$, to be used to compute the *global* edit distance between q and t.
NEEDLEMAN-WUNSH Algorithm:

```
lq = strlen(q); lt = strlen(t);
for( i=0 ; i<=lq ; i++ ) F[i][0] = -i;
for( j=0 ; j<=lt ; j++ ) F[0][j] = -j;

for( i=1 ; i<=lq ; i++ )
    for( j=1 ; j<=lt ; j++ )
        F[i][j] = max3(
            F[i-1][j] -1,   % insertion
            F[i][j-1] -1,   % deletion
            % match/miss match
            F[i-1][j-1] + equal(q[i-1], t[j-1]));
```

`equal()` returns +1 if equal, -1 otherwise.

The value `F[lq][lt]` is the minimum number of edits between q and t. (But what is an edit?)

This method is known for historical reasons as dynamic programming.

Edit distances

The *local* edit distance is similar.
SMITH-WATERMAN algorithm:

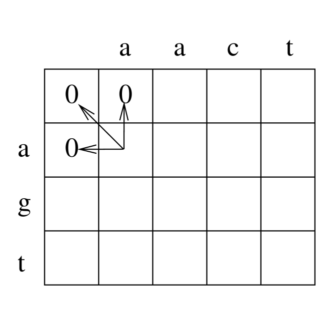
```
lq = strlen(q); lt = strlen(t);
for( i=0 ; i<=lq ; i++ ) F[i][0] = 0;
for( j=0 ; j<=lt ; j++ ) F[0][j] = 0;

for( i=1 ; i<=lq ; i++ )
    for( j=1 ; j<=lt ; j++ )
        F[i][j] = max4(
                        0,
                        F[i-1][j] - 1, % insertion
                        F[i][j-1] - 1, % deletion
                        match/miss match
                        F[i-1][j-1] + equal(q[i-1], t[j-1])
                        );
```

Here, `equal()` returns $+1$ if equal, -1 otherwise. (But there are many choices of scoring schemes, as we discuss later.)

The value of `F[i][j]` with the *highest* value represents the best alignment.

Edit distances

There are three possibilities in the alignment of the first two characters in each sequence: $\begin{bmatrix} \text{a} \\ \text{a} \end{bmatrix}$, $\begin{bmatrix} \text{a} \\ - \end{bmatrix}$, and $\begin{bmatrix} - \\ \text{a} \end{bmatrix}$.



Edit distances

Since $+1$, using alignment $\begin{bmatrix} \text{a} \\ \text{a} \end{bmatrix}$, is the maximum score, we store the score $+1$ in [1, 1] and mark the path used to calculate the optimal score.

		a	a	c	t
	0	0			
a	0	1			
g					
t					

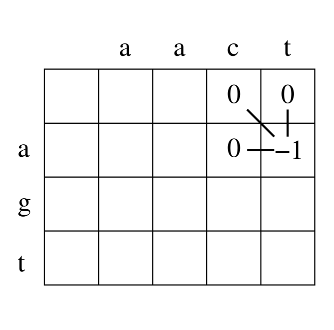
Edit distances

Having considered the first cell, we consider all other cells in the matrix.

		a	a	c	t
	0	0	0	0	0
a	0	1	1	0	−1
g	0	0	0	0	−1
t	0	−1	−1	−1	1

Edit distances

In some cases, such as that of cell [4, 1], the maximum score can be achieved through more than one alignment; in this case, since the scores of cells [4, 0], [3, 0] and [3, 1] are all 0, all paths contribute to a total score in [4, 1] of -1.



Edit distances

The alignment contributing to the maximum score can be reconstructed by tracing-back the pointers stored during the calculation. In this case, there is only one path from [4, 3] that is highlighted with a dashed ellipse: $\begin{bmatrix} \text{aact} \\ \text{–agt} \end{bmatrix}$

		a	a	c	t
	0	0	0	0	0
a	0	1	1	0	−1
g	0	0	0	0	−1
t	0	−1	−1	−1	1

Cost of edit distance

Global alignment of strings was first proposed by Needleman and Wunsch (1970), but is based on at least seven published schemes.

The original formulation of Needleman and Wunsch was $O(n^3)$ in time and $O(n^2)$ in space; their approach considered each cell in row i and each cell in row j in calculating a value for $[i, j]$.

The approach described here is $O(n^2)$ in both time and space.

For short strings, edit distance is practical. For longer strings or comparison against a database of strings, the time cost may be unreasonable.

It isn't obvious how to find strings to test; exhaustive testing is probably infeasible.

Justin Zobel © University of Melbourne, 2011.

N-grams

Let $G_n(s)$ be the substrings of length n of s.

Example1: $G_2(\texttt{Gorbachev})$ is Go, or, rb, ba, ac, ch, he, ev
Example2: $G_2(\texttt{Gorbechyov})$ is Go, or, rb, be, ec, ch, hy, yo, ov

The n-gram distance of t and s is

$$|G_n(s)| + |G_n(t)| - 2 \times |G_n(s) \cap G_n(t)|$$

Example: $s = \texttt{Gorbachev}$, $t = \texttt{Gorbechyov}$, distance is $8 + 9 - 2 \times 4 = 9$.

The smaller the n-gram distance, the closer the match. Cost is approximately $O(|s| + |t|)$, compared to $O(|s| \cdot |t|)$ for an edit distance.

A useful variant, but not much cheaper than an edit distance in practice – it still requires an exhaustive search of the candidates.

Seems unlikely to produce useful scores for long strings or small alphabets.

Aside: Query correction

Web search tools offer corrections to query spellings, typically one or zero alternatives per query.

What is happening under the hood?

Hint: what might the reference material be?

Phonetic matching

Many databases include vocabularies of words or names:

- Surnames in the telephone directory.
- Distinct words occurring in a text database.
- Place names in a gazette.

Phonetic matching techniques allow searching for a name or word when the exact spelling is unknown:

- A user may only know a name by its sound.
- A name can be misspelt in a database.
- Some names have variant or indeterminate spelling.

Example

Georgia Conal

Justin Zobel ©
University of Melbourne, 2011.

Example

Georgia Conal

George O'Connell

Example

Georgia Conal
George O'Connell

Spelling of names is highly variable:

Lho, Lo, Loan, Loe, Loew, Lough, Low, Lowe, . . .

In ≈1 Gb of newspaper articles the name "Gorbachev" had $>$ 20 variants including:

Gorbachev, Gorbacahev, Gorbahev, Gorbatchev, Gorbechev, Gorbachov, Gorachev, Gorbacheva, Gorbechyev, Gorbacev, Gorbachyov, Gorabchev, Grobachev, . . .

Why?

Justin Zobel © University of Melbourne, 2011.

Example

Georgia Conal
George O'Connell

Spelling of names is highly variable:

Lho, Lo, Loan, Loe, Loew, Lough, Low, Lowe, . . .

In $\approx$1 Gb of newspaper articles the name "Gorbachev" had > 20 variants including:

Gorbachev, Gorbacahev, Gorbahev, Gorbatchev, Gorbechev, Gorbachov, Gorachev, Gorbacheva, Gorbechyev, Gorbacev, Gorbachyov, Gorabchev, Grobachev, . . .

Why?

Simple case: Schröder $\rightarrow$ Schroder or Schroeder?

Soundex

The Soundex sound-alike matching technique is well-known, widely used, simple to implement.

Transforms a string into a 4-character code that represents its sound:

- Replace all but the first letter by one of seven single-digit codes:

a e h i o u w y → 0	b f p v → 1
c g j k q s x z → 2	d t → 3
l → 4 m n → 5	r → 6

- Remove doubles, then remove 0s.
- Truncate to four symbols.

Soundex examples

$$\begin{array}{lll} a\,e\,h\,i\,o\,u\,w\,y \rightarrow 0 & & b\,f\,p\,v \rightarrow 1 \\ c\,g\,j\,k\,q\,s\,x\,z \rightarrow 2 & & d\,t \rightarrow 3 \\ l \rightarrow 4 & m\,n \rightarrow 5 & r \rightarrow 6 \end{array}$$

	King	Kyngge
⇒	K052	K05220
⇒	K052	K0520
⇒	K52	K52

	Knight	Night
⇒	K50203	N0203
⇒	K50203	N0203
⇒	K523	N23

	Loan	Loew	Lough	Lewicks
⇒	L005	L000	l002	L000222
⇒	L05	L0	L02	L02
⇒	L5	L	L2	L2

There are 6734 ($26 \times (1 + 6 + 6^2 + 6^3)$) distinct Soundex codes.

There are also (failed) variants, such as Phonix.

Lexicographic methods

Strings that sound alike are often spelt alike.

Example: Gorbachev, Gorbechyov, . . .

It is plausible to suppose that lexicographic string comparison methods will be effective at finding strings that sound alike.

Candidates include neighbourhood search, edit-distances, and n-gram measures.

Perhaps these would be enhanced by combining them with approaches that embody language knowledge.

Refining edit distances

Based on lexicographic information alone, matches such as “Crews” and “Kroose” would not be highly ranked.

Can edit distances be modified to allow for phonetic similarity somehow?

- Repeat letters don’t usually change the sound; “Conel” sounds like “Connell”.
- Sometimes different letters give rise to the same sound; “Kodi” sounds like “Cody”.
- But sometimes they don’t; “like” doesn’t sound like “lice”.

"Editex"

Exploring some possibilities . . . can edit distances be phonetically tweaked?

Zobel's `editex`: Like the standard edit distance, but instead of having a fixed penalty for insertion, deletion, and substitution, have two penalties:

- High for letters that are never similar, such as "d" and "m".
- Low for letters that can give rise to similar sound, such as "m"–"n" and "c"–"k".

Ten groupings:	1. a e i o u y	2. b p	3. c k q	4. d t	5. l r
	6. f p v	7. s x z	8. c s z	9. m n	10. g j

(The silent letters "w" and "h" are handled as a special case.)

Crews–
≀ ≀ |
Kroose

Justin Zobel ©
University of Melbourne, 2011.

But what about phonetics?

Letters don't represent sounds: consider "t" in "trim" versus "think".

"th" represents a sound: the phoneme "Θ" in the International Phonetic Alphabet.

Every name has a pronunciation, or, more precisely, can be represented as a string of phonemes. Perhaps these can be used for matching in some way.

Two problems:

- Deciding what pronunciation corresponds to a given string.
- Comparing pronunciations.

"Ipadist"

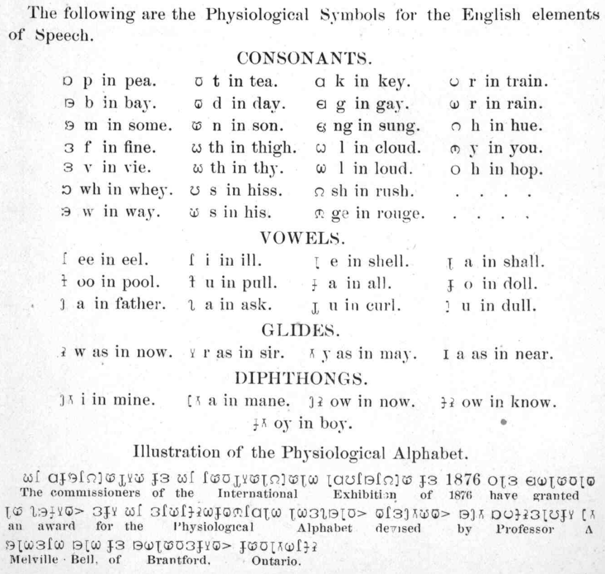
- Use a text-to-sound algorithm to generate the likely pronunciation of each string.
- Use some form of edit distance to compare pronunciations.

Crews	$\rightarrow$	kr j ϱs
Kroose	$\rightarrow$	kr–ϱs

A more principled solution but:

- Text-to-sound algorithms are highly reliable only if context is available, for example to distinguish "in a minute" from "it is minute".
- The pronunciation of names is much less predictable than the pronunciation of words.

Aside: origins of phonetics



The following are the Physiological Symbols for the English elements of Speech.

CONSONANTS.

p in pea.	t in tea.	k in key.	r in train.
b in bay.	d in day.	g in gay.	r in rain.
m in some.	n in son.	ng in sung.	h in hue.
f in fine.	th in thigh.	l in cloud.	y in you.
v in vie.	th in thy.	l in loud.	h in hop.
wh in whey.	s in hiss.	sh in rush.	. . . .
w in way.	s in his.	ge in rouge.	. . . .

VOWELS.

ee in eel.	i in ill.	e in shell.	a in shall.
oo in pool.	u in pull.	a in all.	o in doll.
a in father.	a in ask.	u in curl.	u in dull.

GLIDES.

w as in now.	r as in sir.	y as in may.	a as in near.

DIPHTHONGS.

i in mine.	a in mane.	ow in now.	ow in know.
	oy in boy.		

Illustration of the Physiological Alphabet.

The commissioners of the International Exhibition of 1876 have granted an award for the Physiological Alphabet devised by Professor A Melville Bell, of Brantford, Ontario.

Justin Zobel © University of Melbourne, 2011.

Performance

Measurement of efficiency is straightforward.

But we also need to know whether the method is useful. Measurement of *effectiveness* involves human assessment of the quality of the results.

For each task we need to find an independent method that measures effectiveness in a reliable way.

- Choose a test lexicon and a set of query strings.
- For each query and each word in the lexicon, use relevance assessors to decide whether the word and query match, that is, the word is *relevant* to the query.
- Apply the matching technique to the lexicon and each query to get a list of possible answers, and measure effectiveness by examining how many of the answers are relevant.

Relevance

Applying `agrep -1 davis` to a file of names drawn from internet news postings:

`avis, danis, david, davie, davies, davis,`
`daviss, davys, lavis, navis, ravis`

Using a human to judge which of these matches are probably correct, also considering some other matching techniques and some other names, yields:

barlow	bulow balo barlow
bloom	bloom blume bluhm
clark	klerk kluch clack clerc clarke clark
davis	daviss davyes davys davis
farah	faraj farah farra farrall farrar
hahn	hahm hann hahn hanne

Using relevance information

Given a query, a matching technique A returns a ranked list of answers.

Given relevance judgements, a top-20 ranking from A might be represented as

$$\diamond\ \diamond\ \circ\ \diamond\ \circ\ |\ \circ\ \circ\ \diamond\ \diamond\ \circ\ |\ \circ\ \diamond\ ?\ \diamond\ \circ\ |\ \diamond\ \circ\ ?\ ?\ \circ$$

where $\diamond$ is "correct", $\circ$ is "incorrect", and ? is unjudged. Assuming that unjudged matches are incorrect, this ranking is equivalent to

$$\diamond\ \diamond\ \circ\ \diamond\ \circ\ |\ \circ\ \circ\ \diamond\ \diamond\ \circ\ |\ \circ\ \diamond\ \circ\ \diamond\ \circ\ |\ \diamond\ \circ\ \circ\ \circ\ \circ$$

Justin Zobel ©
University of Melbourne, 2011.

Using relevance information

Reminder: *A*'s ranking is:

$$\diamond\ \diamond\ \circ\ \diamond\ \circ\ |\ \circ\ \circ\ \diamond\ \diamond\ \circ\ |\ \circ\ \diamond\ \circ\ \diamond\ \circ\ |\ \diamond\ \circ\ \circ\ \circ\ \circ$$

Suppose that method *B* returns the ranking

$$\circ\ \diamond\ \diamond\ \diamond\ \diamond\ |\ \circ\ \circ\ \circ\ \diamond\ \circ\ |\ \circ\ \circ\ \diamond\ \circ\ \circ\ |\ \circ\ \diamond\ \circ\ \diamond\ \diamond$$

To decide which method has the greater effectiveness, we need to agree on a method for scoring a ranking.

There is no objective criteria that can be used to assess a scoring method! Choice of measure is based on arguments from properties such as expected user behaviour.

Justin Zobel © University of Melbourne, 2011.

Using relevance information

The traditional measures of effectiveness are precision and recall.

Precision: The proportion of the retrieved matches that are correct.

Reminder – the rankings were:

$$\diamond\ \diamond\ \circ\ \diamond\ \circ \mid \circ\ \circ\ \diamond\ \diamond\ \circ \mid \circ\ \diamond\ \circ\ \diamond\ \circ \mid \diamond\ \circ\ \circ\ \circ\ \circ$$

$$\circ\ \diamond\ \diamond\ \diamond\ \diamond \mid \circ\ \circ\ \circ\ \diamond\ \circ \mid \circ\ \circ\ \diamond\ \circ\ \circ \mid \circ\ \diamond\ \circ\ \diamond\ \diamond$$

In the top 20, *A* has precision 40% (8 out of 20) and *B* has precision 45% (9 out of 20).

The limit of 20 was an arbitrary cut-off; we might instead say, for example, that P@10 for both *A* and *B* is 50%.

Justin Zobel © University of Melbourne, 2011.

Using relevance information

Recall: The proportion of the correct matches that are retrieved.

Reminder – the rankings were:

$$\diamondsuit\ \diamondsuit\ \circ\ \diamondsuit\ \circ\ |\ \circ\ \circ\ \diamondsuit\ \diamondsuit\ \circ\ |\ \circ\ \diamondsuit\ \circ\ \diamondsuit\ \circ\ |\ \diamondsuit\ \circ\ \circ\ \circ\ \circ$$

$$\circ\ \diamondsuit\ \diamondsuit\ \diamondsuit\ \diamondsuit\ |\ \circ\ \circ\ \circ\ \diamondsuit\ \circ\ |\ \circ\ \circ\ \diamondsuit\ \circ\ \circ\ |\ \circ\ \diamondsuit\ \circ\ \diamondsuit\ \diamondsuit$$

We do not know how many correct matches there are for this query. If we assume that 14 strings have been judged correct (not all correct were judged), then A has recall 57% ($8 \div 14$) and B has recall 64% ($9 \div 14$).

Using relevance information

As users tend to proceed down a ranking examining answers, in turn, an alternative is use measures such as average precision (AP) – the average of the precisions at each correct matches retrieved. Correct matches that are not retrieved are assigned a precision of 0.0. AP for *A* is 0.387, and for *B* is 0.363.

Formally, if r_i is 1 (respectively, 0) if the *i*th item in a ranking is relevant (respectively, irrelevant), there are *d* items in the ranking, and there are in total *R* known relevant items, average precision is defined as

$$AP = \frac{1}{R}\sum_{i=1}^{d}\left(\frac{r_i}{i}\cdot\sum_{j=1}^{i} r_j\right)$$

A detailed discussion of this measure is in the readings.

Using relevance information

Different performance measures are not necessarily consistent with each other.

Given the diversity of retrieval methods used by different search engines, it is easy to have $A > B$ for one query and $A < B$ for another. For this reason, we use a sample of queries (10 is too few, 50 is adequate, more is preferable) and average the per-query effectiveness.

We then need to use a statistical test to ensure that the two samples are indeed different (to some level of confidence), rather than simply different by chance.

There are many other measures – F-score, error rate, "receiver operator characteristic" ROC and "area under ROC", discounted cumulative gain, rank-biased precision . . .

The importance of baselines

We use *baselines* to establish whether any proposed method is doing better than "dumb and simple" – "dumb" methods often work surprisingly well.

Baselines are also valuable in getting a sense for the intrinsic difficulty of a given task – sometimes a simple method is so good that there isn't much scope for a clever method to do better.

Example: identifying what language a document is written in. Just need to check which languages have the document's commonest words.

In formulating a baseline, we need to be sensitive to the task and how the method is to be used.
Example:

- In spelling correction it is helpful to give a lot of options; only one of them will be correct.
- In query correction, an option is only shown if there is strong evidence that it is correct.

The baseline should be plausible – there is no point in comparing to the worst method available.

Justin Zobel © University of Melbourne, 2011.

Baselines vs. benchmarks

Baseline – naïve or straightforward method that we would expect a rich (or principled, or . . .) method to do better.

Benchmark – established “best practice” technique that we are pitching our method against.

The word “baseline” is often used as an umbrella term for both meanings.

Building a test collection

Names from the “From:” lines of Internet news articles, hand-edited to standardise format and eliminate obvious rubbish (Darth Vader, GVR 101187-9456, The American Psychological Association).

Queries selected by generating random page numbers in the range 80–1800 and using them as an index into the Melbourne White Pages.

With each of the 125 (!) phonetic matching techniques implemented, fetch the best matches and form a pool.

Using a pair of relevance assessors to judge the pooled answers to each query (one speaking, one listening).

Three teams $\Rightarrow$ three sets of relevance judgements.

Retrieval performance

Average precision (in parentheses, number correct at 200). Each column is a different speaker–assessor team.

Method	Set of judgements		
	A	B	C
Editex	23.1 (17.8)	28.5 (4.3)	26.7 (6.9)
Ipadist	23.2 (16.4)	23.2 (3.8)	24.5 (6.1)
Edit distance	20.4 (15.4)	25.1 (3.7)	22.3 (6.3)
N-gram	20.0 (16.5)	21.5 (3.5)	22.2 (6.2)
Agrep −1	16.5 (3.5)	18.7 (1.2)	18.3 (2.1)
Agrep −best	12.1 (0.8)	20.3 (0.6)	15.2 (0.8)
Soundex	9.3 (6.0)	6.9 (1.8)	9.5 (3.2)

The orderings given by the teams are nearly identical, despite the different scores achieved.

Independence of methods

Consider the query `crews` on the two best methods:

rank	Ipadist	Editex
1.	`crews`	`crews`
2.	`krewe`	`cress`
3.	`kreuser`	`clews`
4.	`crew`	`drews`
5.	`drews`	`crew`
6.	`clews`	`creps`
7.	`kruse`	`kress`
8.	`kroose`	`cross`

That is, measurement based on recall and precision can be similar even if the outcomes are very different.

As mentioned earlier, it is necessary to use a good number of queries to be confident that one method is better than another.

Summary

- Approximate matching techniques provide search with uncertainty.
- Here we have focused on spelling correction, as an example of a simple knowledge technology where effectiveness is important.
- Most methods use some form of distance measure, or do exhaustive search within an error bound.
- For the specific case of phonetic error, special-purpose methods are superior to purely alphabetic methods.
- Testing of methods requires a test data set; test systems or methods, including a baseline; human assessment of outcomes; an effectiveness measure.
- Such testing methods underpin investigation of areas such as text search and document classification.

Readings

`en.wikipedia.org/wiki/Bitap_algorithm`

`en.wikipedia.org/wiki/Spell_checker`

`www.googleguide.com/spelling_corrections.html`

"Learning a Spelling Error Model from Search Query Logs" (LMS).

"Rank-Biased Precision for Measurement of Retrieval Effectiveness" (LMS), sections 2 & 3.

"Phonetic String Matching: Lessons from Information Retrieval" (LMS).

Manning et al., chapters 3 & 8.