Approximate String Search and Matching

COMP90049
COMP30018
Knowledge Technologies

Lea Frermann and Justin Zobel and Karin Verspoor

Semester 2, 2019

THE UNIVERSITY OF MELBOURNE

Roadmap

Last week... Probability and similarity

Nuts and bolts of knowledge technologies!

- quantifying similarity of complex structures
- estimate the (conditional, joint) probability of observations
- identify high entropy (=informative) information

This week... Approximate String Search and Matching

Another bolt!

- back to similarity
- methods, applications, evaluation

Exact String Search

- Given a string, is some substring contained within it?
- Given a string (document), find all occurrences of some substring

Exact String Search

- Given a string, is some substring contained within it?
- Given a string (document), find all occurrences of some substring

For example, find `Exxon` in:

```
In exes for foxes rex dux mixes a pox of waxed luxes.
An axe, and an axon, to exo Exxon max oxen.
Grexit or Brexit as quixotic haxxers with buxom rex taxation.
```

Exact String Search

- Given a string, is some substring contained within it?
- Given a string (document), find all occurrences of some substring

For example, find `Exxon` in:

```
In exes for foxes rex dux mixes a pox of waxed luxes.
An axe, and an axon, to exo Exxon max oxen.
Grexit or Brexit as quixotic haxxers with buxom rex taxation.
```

Exact String Search

- Given a string, is some substring contained within it?
- Given a string (document), find all occurrences of some substring

For example, find `Exxon` in:

```
In exes for foxes rex dux mixes a pox of waxed luxes.
An axe, and an axon, to exo Exxon max oxen.
Grexit or Brexit as quixotic haxxers with buxom rex taxation.
```

Not (really) a Knowledge Technology!

Approximate String Search

Find `exon` in:

```
In exes for foxes rex dux mixes a pox of waxed luxes.
An axe, and an axon, to exo Exxon max oxen.
Grexit or Brexit as quixotic haxxers with buxom rex taxation.
```

Approximate String Search

Find `exon` in:

```
In exes for foxes rex dux mixes a pox of waxed luxes.
An axe, and an axon, to exo Exxon max oxen.
Grexit or Brexit as quixotic haxxers with buxom rex taxation.
```

Not present!

...But what is the "closest" or "best" match?

Approximate String Search

Find `exon` in:

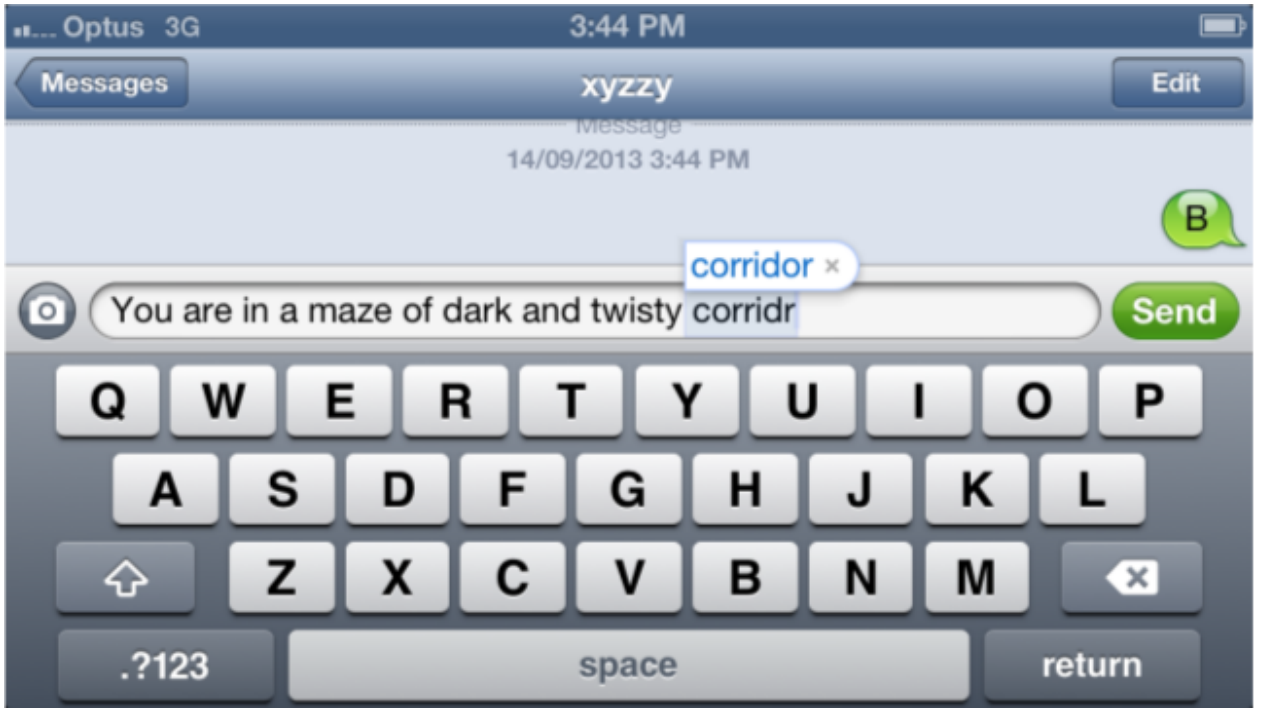
```
In exes for foxes rex dux mixes a pox of waxed luxes.
An axe, and an axon, to exo Exxon max oxen.
Grexit or Brexit as quixotic haxxers with buxom rex taxation.
```

Not present!

...But what is the "closest" or "best" match?

This is a Knowledge Technology!

Spelling Correction



Spelling Correction

Need the notion of a **dictionary**:

- Here, a list of words

Spelling Correction

Need the notion of a **dictionary**:

- Here, a list of ~~words~~ entries that are "correct" with respect to our (expectations of our) language

Spelling Correction

Need the notion of a **dictionary**:

- Here, a list of ~~words~~ entries that are "correct" with respect to our (expectations of our) language
- We can break our input into ~~words~~ substrings that we wish to match, and compare each of them against the entries in the dictionary

Spelling Correction

Need the notion of a **dictionary**:

- Here, a list of ~~words~~ entries that are "correct" with respect to our (expectations of our) language
- We can break our input into ~~words~~ substrings that we wish to match, and compare each of them against the entries in the dictionary
- A ~~word~~ item in the input which *doesn't* appear in the dictionary is *misspelled*

Spelling Correction

Need the notion of a **dictionary**:

- Here, a list of ~~words~~ entries that are "correct" with respect to our (expectations of our) language
- We can break our input into ~~words~~ substrings that we wish to match, and compare each of them against the entries in the dictionary
- A ~~word~~ item in the input which *doesn't* appear in the dictionary is *misspelled*
- A ~~word~~ item in the input which *does* appear in the dictionary might be correctly spelled *or* misspelled (probably slightly beyond the scope of this subject)

Spelling Correction

Therefore, the problem here:

Given some item of interest — which does not appear in our dictionary — which entry from the dictionary was truly intended?

Spelling Correction

Therefore, the problem here:

Given some item of interest — which does not appear in our dictionary — which entry from the dictionary was truly intended?

Depends on the person who wrote the original string!

Detecting novel words

Word Blending:

- forming novel words by "blending" two other words
- not simple concatenation (e.g., football is not a blend word)

breakfast	+	lunch	$\rightarrow$	brunch
fork	+	spoon	$\rightarrow$	spork
Britain	+	exit	$\rightarrow$	Brexit

Detecting novel words

Word Blending:

- forming novel words by "blending" two other words
- not simple concatenation (e.g., football is not a blend word)

breakfast	+	lunch	$\rightarrow$	brunch
fork	+	spoon	$\rightarrow$	spork
Britain	+	exit	$\rightarrow$	Brexit

- Language changes constantly!
- New terms often coined in colloquial language (e.g., Twitter)
- Can we build **knowledge technologies** that detect novel blends? ...or help us understand what their **components** are?
- **Assignment 1**

Other Problems of Interest

Name matching

the name *Gorbachev* is spelled (at least) 20 different ways in a corpus of newswire text!

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

street and place name conventions

`boulevard|blvd|bd|bde|blv|bl|blvde|blvrd|boulavard|boul|bvd`
`apartment|apt|ap|aprt|aptmnt`
`village|vil|vge|vill|villag|villg|vlg|vlge|vllg`

Other Problems of Interest

- Computational Genomics
- Name matching
- Query repair
- Phonetic matching
- Data cleaning (e.g., deduplication)
- ...

Aside: Text Normalization

Before we start... fix some irregularities which may distract the matching algorithm.

Aside: Text Normalization

Original: `I had an AMAZING trip to Italy, Coffee is only 2 bucks, sometimes three! it's increadible!!! Right?`

Aside: Text Normalization

Original: `I had an AMAZING trip to Italy, Coffee is only 2 bucks, sometimes three! it's increadible!!! Right?`

Normalized: `i had an amazing trip to italy, coffee is only 2 bucks, sometimes three! it's increadible!!! right?`

1. Consistent Casing

map all characters to upper (or lower) case

Aside: Text Normalization

Original: `I had an AMAZING trip to Italy, Coffee is only 2 bucks, sometimes three! it's increadible!!! Right?`

Normalized: `i had an amazing trip to italy, coffee is only two bucks, sometimes three! it's increadible!!! right?`

2. Unify or remove numbers

map numbers to a consistent representation or remove (replace) them entirely

Aside: Text Normalization

Original: `I had an AMAZING trip to Italy, Coffee is only 2 bucks, sometimes three! it's increadible!!! Right?`

Normalized: `i had an amazing trip to italy, coffee is only two bucks, sometimes three! it's increadible!!! right?`

3. Remove unnecessary spacing

e.g., duplicate white spaces

Aside: Text Normalization

Original: `I had an AMAZING trip to Italy, Coffee is only 2 bucks, sometimes three! it's increadible!!! Right?`

Normalized: `i had an amazing trip to italy , coffee is only two bucks , sometimes three ! it 's increadible ! ! ! Right ?`

4. Word Tokenization

split text into individual tokens (aka words)

Aside: Text Normalization

Original: `I had an AMAZING trip to Italy, Coffee is only 2 bucks, sometimes three! it's increadible!!! Right?`

Normalized: `i had an amazing trip to italy , coffee is only two bucks , sometimes three ! it 's` **increadible** `! ! ! Right ?`

4. Word Tokenization

split text into individual tokens (aka words)

What's a "best" match?

Find approximate match(es) for `exon` in:

```
In exes for foxes rex dux mixes a pox of waxed luxes.
An axe, and an axon, to exo Exxon max oxen.
Grexit or Brexit as quixotic haxxers with buxom rex taxation.
```

What's a "best" match?

Find approximate match(es) for `exon` in:

`In exes for foxes rex dux mixes a pox of waxed luxes.`
`An axe, and an axon, to exo` **Exxon** `max oxen.`
`Grexit or Brexit as quixotic haxxers with buxom rex taxation.`

`exon` →	Exxon	**Insert** `x`
`exon` →	exo	**Delete** `n`
`exon` →	axon	**Replace** `e` with `a` (Sometimes **Substitute**)
`exon` →	oxen	**Transpose** `e` and `o` (not covered here)

What's a "best" match?

Find approximate match(es) for `exon` in:

```
In exes for foxes rex dux mixes a pox of waxed luxes.
An axe, and an axon, to exo Exxon max oxen.
Grexit or Brexit as quixotic haxxers with buxom rex taxation.
```

`exon` →	Exxon	**Insert** `x`
`exon` →	exo	**Delete** `n`
`exon` →	axon	**Replace** `e` with `a` (Sometimes **Substitute**)
`exon` →	oxen	**Transpose** `e` and `o` (not covered here)

What's a "best" match?

Find approximate match(es) for `exon` in:

```
In exes for foxes rex dux mixes a pox of waxed luxes.
An axe, and an axon , to exo Exxon max oxen.
Grexit or Brexit as quixotic haxxers with buxom rex taxation.
```

`exon` →	Exxon	**Insert** `x`
`exon` →	exo	**Delete** `n`
`exon` →	axon	**Replace** `e` with `a` (Sometimes **Substitute**)
`exon` →	oxen	**Transpose** `e` and `o` (not covered here)

What's a "best" match?

Find approximate match(es) for `exon` in:

```
In exes for foxes rex dux mixes a pox of waxed luxes.
An axe, and an axon, to exo Exxon max oxen.
Grexit or Brexit as quixotic haxxers with buxom rex taxation.
```

`exon` →	Exxon	**Insert** `x`
`exon` →	exo	**Delete** `n`
`exon` →	axon	**Replace** `e` with `a` (Sometimes **Substitute**)
`exon` →	oxen	**Transpose** `e` and `o` (not covered here)

Neighbourhood Search

For a given string *w* of interest:

- Generate all variants of *w* that utilise at most *k* changes (Insertions/Deletions/Replacements) — **neighbours**

Neighbourhood Search

For a given string *w* of interest:

- Generate all variants of *w* that utilise at most *k* changes (Insertions/Deletions/Replacements) — **neighbours**
- Check whether generated variants exist in dictionary

Neighbourhood Search

For a given string *w* of interest:

- Generate all variants of *w* that utilise at most *k* changes (Insertions/Deletions/Replacements) — **neighbours**
- Check whether generated variants exist in dictionary
- **All** results found in dictionary are returned

Neighbourhood Search

For a given string *w* of interest:

- Generate all variants of *w* that utilise at most *k* changes (Insertions/Deletions/Replacements) — **neighbours**
- Check whether generated variants exist in dictionary
- **All** results found in dictionary are returned

Unix command-line utility `agrep` is an efficient tool for finding these.

Neighbourhood Search

For a given string *w* of interest:

- Generate all variants of *w* that utilise at most *k* changes (Insertions/Deletions/Replacements) — **neighbours**
- Check whether generated variants exist in dictionary
- **All** results found in dictionary are returned

Unix command-line utility `agrep` is an efficient tool for finding these.

For example:

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

Neighbourhood Search

For a given string *w* of interest:

- Generate all variants of *w* that utilise at most *k* changes (Insertions/Deletions/Replacements) — **neighbours**
- Check whether generated variants exist in dictionary
- **All** results found in dictionary are returned

Unix command-line utility `agrep` is an efficient tool for finding these.

For example:

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

the their there tier **other** mther tnher thpr ...

Neighborhood search

Neighbourhood Search Efficiency

Neighborhood search

Consider: alphabet size is Σ, length of string is $|w|$:

For 1 edit, roughly $\mathcal{O}(\Sigma \cdot |w|)$ neighbours

Neighbourhood Search Efficiency

Neighborhood search

Consider: alphabet size is Σ, length of string is $|w|$:

For 2 edits, roughly $\mathcal{O}(\Sigma^2 \cdot |w|^2)$ neighbours

Neighbourhood Search Efficiency

Neighborhood search

Consider: alphabet size is Σ, length of string is $|w|$:

For k edits, roughly $\mathcal{O}(\Sigma^k \cdot |w|^k)$ neighbours

Neighbourhood Search Efficiency

Neighborhood search

Consider: alphabet size is Σ, length of string is $|w|$:

For k edits, roughly $\mathcal{O}(\Sigma^k \cdot |w|^k)$ neighbours

But,

- Σ is a small constant
- string of interest is usually short
- k is usually small

Neighbourhood Search Efficiency

Neighborhood search

Consider: alphabet size is Σ, length of string is $|w|$:

For k edits, roughly $\mathcal{O}(\Sigma^k \cdot |w|^k)$ neighbours

But,

- Σ is a small constant
- string of interest is usually short
- k is usually small

Dictionary Read

Assuming D entries, binary search yields $\mathcal{O}(|w|^k \log D)$ string comparisons

Neighbourhood Search Efficiency

Neighborhood search

Consider: alphabet size is Σ, length of string is $|w|$:

For k edits, roughly $\mathcal{O}(\Sigma^k \cdot |w|^k)$ neighbours

But,

- Σ is a small constant
- string of interest is usually short
- k is usually small

`agrep` example

Dictionary Read

Assuming D entries, binary search yields $\mathcal{O}(|w|^k \log D)$ string comparisons

Edit Distance

An alternative method:

Scan through each dictionary entry looking for the "best" match

Edit Distance

An alternative method:

Scan through each dictionary entry looking for the "best" match

Intuition:

- Transform the string of interest into each dictionary entry
- **Operations**: Insert, Delete, Replace, and Match

Edit Distance

An alternative method:

Scan through each dictionary entry looking for the "best" match

Intuition:

- Transform the string of interest into each dictionary entry
- **Operations**: Insert, Delete, Replace, and Match
- Each operation is associated with a score;

Edit Distance

An alternative method:

Scan through each dictionary entry looking for the "best" match

Intuition:

- Transform the string of interest into each dictionary entry
- **Operations**: Insert, Delete, Replace, and Match
- Each operation is associated with a score;
- Best match is the dictionary entry with best aggregate **score**

Edit Distance

An alternative method:

Scan through each dictionary entry looking for the "best" match

Intuition:

- Transform the string of interest into each dictionary entry
- **Operations**: Insert, Delete, Replace, and Match
- Each operation is associated with a score;
- Best match is the dictionary entry with best aggregate **score**

Global Edit Distance – Example

Item of interest	`crat`
Dictionary	`cart`, `arts`
Scores	Match +1, Insert -1, Delete -1, Replace -1

1. Transform `crat` $\rightarrow$ `cart`

Global Edit Distance – Example

Item of interest	`crat`
Dictionary	`cart`, `arts`
Scores	Match +1, Insert -1, Delete -1, Replace -1

1. Transform `crat` $\rightarrow$ `cart`

   Match `c` Delete `r` Match `a` Insert `r` Match `t`

Global Edit Distance – Example

Item of interest	`crat`
Dictionary	`cart`, `arts`
Scores	Match +1, Insert -1, Delete -1, Replace -1

1. Transform `crat` $\rightarrow$ `cart`

Match `c`	Delete `r`	Match `a`	Insert `r`	Match `t`	
+1	-1	+1	-1	+1	= +1

Global Edit Distance – Example

Item of interest	`crat`
Dictionary	`cart`, `arts`
Scores	Match +1, Insert -1, Delete -1, Replace -1

1. Transform `crat` → `cart`

Match `c`	Delete `r`	Match `a`	Insert `r`	Match `t`	
+1	-1	+1	-1	+1	= +1

2. Transform `crat` → `arts`

Global Edit Distance – Example

Item of interest	`crat`
Dictionary	`cart`, `arts`
Scores	Match +1, Insert -1, Delete -1, Replace -1

1. Transform `crat` → `cart`

Match `c`	Delete `r`	Match `a`	Insert `r`	Match `t`	
+1	-1	+1	-1	+1	= +1

2. Transform `crat` → `arts`

Replace `c`, `a` Match `r` Delete `a` Match `t` Insert s

Global Edit Distance – Example

Item of interest	`crat`
Dictionary	`cart`, `arts`
Scores	Match +1, Insert -1, Delete -1, Replace -1

1. Transform `crat` → `cart`

Match `c`	Delete `r`	Match `a`	Insert `r`	Match `t`	
+1	-1	+1	-1	+1	= +1

2. Transform `crat` → `arts`

Replace `c`, `a`	Match `r`	Delete `a`	Match `t`	Insert s	
-1	+1	-1	+1	-1	= -1

Global Edit Distance – Example

Item of interest	`crat`
Dictionary	`cart`, `arts`
Scores	Match +1, Insert -1, Delete -1, Replace -1

1. Transform `crat` → `cart`

Match `c`	Delete `r`	Match `a`	Insert `r`	Match `t`	
+1	-1	+1	-1	+1	= +1

2. Transform `crat` → `arts`

Replace `c`, `a`	Match `r`	Delete `a`	Match `t`	Insert s	
-1	+1	-1	+1	-1	= -1

`cart` is the better match

Global Edit Distance – Choice of Parameter

Confusingly, Global Edit Distance isn't necessarily a "distance"

Why?

Global Edit Distance – Choice of Parameter

Confusingly, Global Edit Distance isn't necessarily a "distance"

Why?

But we can make it one

- Global edit distance with parameters:

  Match (0), Insert (+1), Delete (+1), Replace (+1)

- counts the number of edits required to transform one string into the other
- **Levenshtein Distance**

Global Edit Distance – Choice of Parameter

Confusingly, Global Edit Distance isn't necessarily a "distance"

Why?

But we can make it one

- Global edit distance with parameters:

  Match (0), Insert (+1), Delete (+1), Replace (+1)
- counts the number of edits required to transform one string into the other
- **Levenshtein Distance**

Often,

- cost(Insert) = cost(Delete) (direction doesn't matter)

Global Edit Distance – Choice of Parameter

Confusingly, Global Edit Distance isn't necessarily a "distance"

Why?

But we can make it one

- Global edit distance with parameters:

  Match (0), Insert (+1), Delete (+1), Replace (+1)
- counts the number of edits required to transform one string into the other
- **Levenshtein Distance**

Often,

- cost(Insert) = cost(Delete) (direction doesn't matter)
- score of Replace depends on the character being replaced

Is `faxing` more likely to be `facing` or `faking`?

Global Edit Distance – Example

Item of interest	`crat`
Dictionary	`cart`, `arts`
Scores	Match 0, Insert +1, Delete +1, Replace +1

1. Transform `crat` → `cart`

Match `c`	Delete `r`	Match `a`	Insert `r`	Match `t`	
0	+1	0	+1	0	= +2

2. Transform `crat` → `arts`

Replace `c`, `a`	Match `r`	Delete `a`	Match `t`	Insert s	
+1	0	+1	0	+1	= +3

3 changes vs 2 changes: `cart` is the better match

Global Edit Distance – Choice of Parameter

Hypothetically, any parameter is possible!

But some choices make no sense, e.g.:

Match (+4), Insert (-2), Delete (+8), Replace (0)

Consider `aba`: which corresponds to best match?

- `foo`
- `aba`
- `cbc`

Global Edit Distance – Choice of Parameter

Hypothetically, any parameter is possible!

But some choices make no sense, e.g.:

Match (+4), Insert (-2), Delete (+8), Replace (0)

Consider `aba`: which corresponds to best match?

- `foo`: Insert, Delete, Insert, Delete, Insert, Delete
- `aba`: Match, Match, Match
- `cbc`: Replace, Match, Replace

Global Edit Distance – Choice of Parameter

Hypothetically, any parameter is possible!

But some choices make no sense, e.g.:

Match (+4), Insert (-2), Delete (+8), Replace (0)

Consider `aba`: which corresponds to best match?

- `foo`: Insert, Delete, Insert, Delete, Insert, Delete = +18
- `aba`: Match, Match, Match = +12
- `cbc`: Replace, Match, Replace = +4

Global Edit Distance Algorithm

Computer can't find best sequence of operations by inspection

Global Edit Distance Algorithm

From string f to string t, given array A of $|f| + 1$ columns and $|t| + 1$ rows, we can solve using the Needleman–Wunsch algorithm:

```
lf = strlen(f); lt = strlen(t);
A[0][0]=0;
for (j=1; j<=lt; j++) A[j][0] = j * i;
for (k=1; k<=lf; k++) A[0][k] = k * d;

for (j=1; j<=lt; j++)
   for (k=1; k<=lf; k++)
      A[j][k] = max3( //Or min3 if m<i,d,r
         A[j][k-1] + d, //Deletion
         A[j-1][k] + i, //Insertion
         A[j-1][k-1] + equal(f[k-1],t[j-1])); //Replace or match
```

`equal()` returns m if characters match, r otherwise

Final score is at $A[lt][lf]$

Global Edit Distance in Action

In action: from `crat` to `arts`, Match (+1), Insert/Delete/Replace (-1)

$\rightarrow$ delete source word
$\downarrow$ insert target word

Global Edit Distance in Action

In action: from `crat` to `arts`, Match (+1), Insert/Delete/Replace (-1)

$\rightarrow$ delete source word
$\downarrow$ insert target word

	ε	c	r	a	t
ε					
a					
r					
t					
s					

Global Edit Distance in Action

In action: from `crat` to `arts`, Match (+1), Insert/Delete/Replace (-1)

$\rightarrow$ delete source word
$\downarrow$ insert target word

Initialise table:

	ε	c	r	a	t
ε	0	-1	-2	-3	-4
a	-1				
r	-2				
t	-3				
s	-4				

Global Edit Distance in Action

In action: from `crat` to `arts`, Match (+1), Insert/Delete/Replace (-1)

$\rightarrow$ delete source word
$\downarrow$ insert target word

For `c`–`a` correspondence, consider three neighbours:

	ε	c	r	a	t
ε	0	-1	-2	-3	-4
a	-1	?			
r	-2				
t	-3				
s	-4				

Global Edit Distance in Action

In action: from `crat` to `arts`, Match (+1), Insert/Delete/Replace (-1)

$\rightarrow$ delete source word
$\downarrow$ insert target word

For `c`–`a` correspondence, Delete `c`:

	ε	c	r	a	t
ε	0	-1	-2	-3	-4
a	-1	-2			
r	-2				
t	-3				
s	-4				

Global Edit Distance in Action

In action: from `crat` to `arts`, Match (+1), Insert/Delete/Replace (-1)

$\rightarrow$ delete source word
$\downarrow$ insert target word

For `c`–`a` correspondence, Insert `a`:

	ε	c	r	a	t
ε	0	-1	-2	-3	-4
a	-1	-2			
r	-2				
t	-3				
s	-4				

Global Edit Distance in Action

In action: from `crat` to `arts`, Match (+1), Insert/Delete/Replace (-1)

$\rightarrow$ delete source word
$\downarrow$ insert target word

For `c`–`a` correspondence, Replace `c` with `a`:

	ε	c	r	a	t
ε	0	-1	-2	-3	-4
a	-1	-1			
r	-2				
t	-3				
s	-4				

Global Edit Distance in Action

In action: from `crat` to `arts`, Match (+1), Insert/Delete/Replace (-1)

$\rightarrow$ delete source word
$\downarrow$ insert target word

And so on:

	ε	c	r	a	t
ε	0	-1	-2	-3	-4
a	-1	-1	-2		
r	-2				
t	-3				
s	-4				

Global Edit Distance in Action

In action: from `crat` to `arts`, Match (+1), Insert/Delete/Replace (-1)

$\rightarrow$ delete source word
$\downarrow$ insert target word

And so on:

	ε	c	r	a	t
ε	0	-1	-2	-3	-4
a	-1	-1	-2	-1	
r	-2				
t	-3				
s	-4				

Global Edit Distance in Action

In action: from `crat` to `arts`, Match (+1), Insert/Delete/Replace (-1)

$\rightarrow$ delete source word
$\downarrow$ insert target word

And so on:

	ε	c	r	a	t
ε	0	-1	-2	-3	-4
a	-1	-1	-2	-1	-2
r	-2	-2	0	-1	-2
t	-3	-3	-1	-1	0
s	-4	-4	-2	-2	-1

Global Edit Distance in Action

In action: from `crat` to `arts`, Match (+1), Insert/Delete/Replace (-1)

$\rightarrow$ delete source word
$\downarrow$ insert target word

And so on:

	ε	c	r	a	t
ε	0	-1	-2	-3	-4
a	-1	-1	-2	-1	-2
r	-2	-2	0	-1	-2
t	-3	-3	-1	-1	0
s	-4	-4	-2	-2	-1

Global Edit Distance: -1 (Replace, Match, Delete, Match, Insert)

More parameter concerns

Algorithm actually depends on parameter!

More parameter concerns

Algorithm actually depends on parameter!

```
A[j][k] = max3(
  A[j][k-1] + d, //Deletion
  A[j-1][k] + i, //Insertion
  A[j-1][k-1] + equal(f[k-1],t[j-1])); //Replace or match
```

More parameter concerns

Algorithm actually depends on parameter!

```
A[j][k] = max3(
  A[j][k-1] + d, //Deletion
  A[j-1][k] + i, //Insertion
  A[j-1][k-1] + equal(f[k-1],t[j-1])); //Replace or match
```

→ Match score greater than Insert/Delete/Replace

e.g. Match (+1), Insert/Delete/Replace (-1)

More parameter concerns

Algorithm actually depends on parameter!

```
A[j][k] = min3(
  A[j][k-1] + d, //Deletion
  A[j-1][k] + i, //Insertion
  A[j-1][k-1] + equal(f[k-1],t[j-1])); //Replace or match
```

More parameter concerns

Algorithm actually depends on parameter!

```
A[j][k] = min3(
  A[j][k-1] + d, //Deletion
  A[j-1][k] + i, //Insertion
  A[j-1][k-1] + equal(f[k-1],t[j-1])); //Replace or match
```

$\rightarrow$ Match score less than Insert/Delete/Replace

e.g. Match (0), Insert/Delete/Replace (+1)

(Levenshtein Distance)

Local Edit Distance

like Global Edit Distance, but we are searching for the best substring match

Local Edit Distance

like Global Edit Distance, but we are searching for the best substring match

Particularly suitable when comparing two strings of very different lengths, e.g.

- a suffix of a word
- a word in a sentence
- a sentence in an entire document

Local Edit Distance Algorithm

From string f to string t, given array A of $|f| + 1$ columns and $|t| + 1$ rows, we can solve using the Smith–Waterman algorithm:

```
lf = strlen(f); lt = strlen(t);
A[0][0]=0;
for (j=1; j<=lt; j++) A[j][0] = 0;
for (k=1; k<=lf; k++) A[0][k] = 0;

for (j=1; j<=lt; j++)
   for (k=1; k<=lf; k++)
      A[j][k] = max4( //Or min4 if m<i,d,r
         0,
         A[j][k-1] + d, //Deletion
         A[j-1][k] + i, //Insertion
         A[j-1][k-1] + equal(f[k-1],t[j-1])); //Replace or match
```

`equal()` returns m if characters match, r otherwise

Final score is greatest value in the entire table (or least value, if $m < i, d, r$)

Local Edit Distance in Action

In action: from `cart` to `arts`, Match (+1), Insert/Delete/Replace (-1)

$\rightarrow$ delete source word
$\downarrow$ insert target word

(For Local Edit Distance, Match must have different $+/-$ sign to Insert/Delete/Replace)

Local Edit Distance in Action

In action: from `cart` to `arts`, Match (+1), Insert/Delete/Replace (-1)

$\rightarrow$ delete source word
$\downarrow$ insert target word

	ε	c	a	r	t
ε					
a					
r					
t					
s					

Local Edit Distance in Action

In action: from `cart` to `arts`, Match (+1), Insert/Delete/Replace (-1)

$\rightarrow$ delete source word
$\downarrow$ insert target word

Initialise table:

	ε	c	a	r	t
ε	0	0	0	0	0
a	0				
r	0				
t	0				
s	0				

Local Edit Distance in Action

In action: from `cart` to `arts`, Match (+1), Insert/Delete/Replace (-1)

$\rightarrow$ delete source word
$\downarrow$ insert target word

For `c`–`a` correspondence, consider three neighbours:

	ε	c	a	r	t
ε	0	0	0	0	0
a	0	?			
r	0				
t	0				
s	0				

Local Edit Distance in Action

In action: from `cart` to `arts`, Match (+1), Insert/Delete/Replace (-1)

→ delete source word
↓ insert target word

For `c`–`a` correspondence, Delete `c`:

	ε	c	a	r	t
ε	0	0	0	0	0
a	0	-1			
r	0				
t	0				
s	0				

Local Edit Distance in Action

In action: from `cart` to `arts`, Match (+1), Insert/Delete/Replace (-1)

$\rightarrow$ delete source word
$\downarrow$ insert target word

For `c`–`a` correspondence, Insert `a`:

	ε	c	a	r	t
ε	0	0	0	0	0
a	0	-1			
r	0				
t	0				
s	0				

Local Edit Distance in Action

In action: from `cart` to `arts`, Match (+1), Insert/Delete/Replace (-1)

$\rightarrow$ delete source word
$\downarrow$ insert target word

For `c`–`a` correspondence, Replace `c` with `a`:

	ε	c	a	r	t
ε	0	0	0	0	0
a	0	-1			
r	0				
t	0				
s	0				

Local Edit Distance in Action

In action: from `cart` to `arts`, Match (+1), Insert/Delete/Replace (-1)

$\rightarrow$ delete source word
$\downarrow$ insert target word

For `c`–`a` correspondence, 0 is better:

	ε	c	a	r	t
ε	0	0	0	0	0
a	0	0			
r	0				
t	0				
s	0				

Local Edit Distance in Action

In action: from `cart` to `arts`, Match (+1), Insert/Delete/Replace (-1)

$\rightarrow$ delete source word
$\downarrow$ insert target word

For `a`–`a` correspondence (Match), 1 is better:

	ε	c	a	r	t
ε	0	0	0	0	0
a	0	0	1		
r	0				
t	0				
s	0				

Local Edit Distance in Action

In action: from `cart` to `arts`, Match (+1), Insert/Delete/Replace (-1)

$\rightarrow$ delete source word
$\downarrow$ insert target word

For `a`–`r` correspondence, back to 0:

	ε	c	a	r	t
ε	0	0	0	0	0
a	0	0	1	0	
r	0				
t	0				
s	0				

Local Edit Distance in Action

In action: from `cart` to `arts`, Match (+1), Insert/Delete/Replace (-1)

$\rightarrow$ delete source word
$\downarrow$ insert target word

And so on:

	ε	c	a	r	t
ε	0	0	0	0	0
a	0	0	1	0	0
r	0	0	0	2	1
t	0	0	0	1	3
s	0	0	0	0	2

Local Edit Distance in Action

In action: from `cart` to `arts`, Match (+1), Insert/Delete/Replace (-1)

$\rightarrow$ delete source word
$\downarrow$ insert target word

And so on:

	ε	c	a	r	t
ε	0	0	0	0	0
a	0	0	1	0	0
r	0	0	0	2	1
t	0	0	0	1	3
s	0	0	0	0	2

Best match: `art` with `art` (+3); ties are possible.

Edit Distance Efficiency

- single source string f
- multiple target strings t in dictionary D

- a single approximate match (both global and local)

$$\mathcal{O}(|f||t|)$$

Edit Distance Efficiency

- single source string f
- multiple target strings t in dictionary D

- a single approximate match (both global and local)

$$\mathcal{O}(|f||t|)$$

- approximate matching of f over the whole dictionary

$$\mathcal{O}(\sum_{t \in D} |f||t|)$$

Edit Distance Efficiency

- single source string f
- multiple target strings t in dictionary D

- a single approximate match (both global and local)

$$\mathcal{O}(|f||t|)$$

- approximate matching of f over the whole dictionary

$$\mathcal{O}\left(\sum_{t \in D} |f||t|\right) = \mathcal{O}\left(|f| \sum_{t \in D} |t|\right)$$

Edit Distance Efficiency

- single source string f
- multiple target strings t in dictionary D

- a single approximate match (both global and local)

$$\mathcal{O}(|f||t|)$$

- approximate matching of f over the whole dictionary

$$\mathcal{O}\left(\sum_{t \in D} |f||t|\right) = \mathcal{O}\left(|f| \sum_{t \in D} |t|\right)$$

- **feasibility depends (linearly) on size of the dictionary**

Background Readings

Needleman, Saul B. and Wunsch, Christian D. (1970). "A general method applicable to the search for similarities in the amino acid sequence of two proteins". Journal of Molecular Biology 48 (3): 443–53. doi:10.1016/0022-2836(70)90057-4

(Originally in Russian, published in English as:) Levenshtein, Vladimir I. (1966). "Binary codes capable of correcting deletions, insertions, and reversals". Soviet Physics Doklady 10 (8): 707–710.

Christin, P. (2006). "A Comparison of Personal Name Matching: Techniques and Practical Issues". Proceedings of the Sixth IEEE International Conference on Data Mining - Workshops

Background Readings

Kondrak, Grzegorz (2005). "N-Gram Similarity and Distance". In Proceedings of the 12th international conference on String Processing and Information Retrieval (SPIRE'05), pp. 115-126, Buenos Aires, Argentina.

Peng, N. and Yu, M. and Drezde, M. (2015). "An Empirical Study of Chinese Name Matching and Applications". In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers), pp. 377–383, Bejing, China.

Extension Readings

Whitelaw, Casey and Hutchison, Ben and Chung, Grace Y and Ellis, Gerard (2009). "Using the Web for Language Independent Spellchecking and Autocorrection". In Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP 2009), pp. 890-899, Singapore, Singapore.

Ahmad, Farooq and Kondrak, Grzegorz (2005). "Learning a Spelling Error Model from Search Query Logs". In Proceedings of the Human Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP 2005), pp. 955-962, Vancouver, Canada.

Soundex

One (ineffectual) mechanism: Soundex

Soundex

One mechanism: Soundex

Translation table:

`aehiouwy`	→	0 (vowels)
`bpfv`	→	1 (labials)
`cgjkqsxz`	→	2 (misc: fricatives, velars, etc.)
`dt`	→	3 (dentals)
`l`	→	4 (lateral)
`mn`	→	5 (nasals)
`r`	→	6 (rhotic)

Soundex

One mechanism: Soundex

Translation table:

`aehiouwy`	→	0 (vowels)
`bpfv`	→	1 (labials)
`cgjkqsxz`	→	2 (misc: fricatives, velars, etc.)
`dt`	→	3 (dentals)
`l`	→	4 (lateral)
`mn`	→	5 (nasals)
`r`	→	6 (rhotic)

Four step process:

1. Except for initial character, translate string characters according to table
2. Remove duplicates (e.g. `4444` → `4`)
3. Remove 0s
4. Truncate to four symbols

Soundex

One mechanism: Soundex

Translation table:

`aehiouwy`	→	0 (vowels)
`bpfv`	→	1 (labials)
`cgjkqsxz`	→	2 (misc: fricatives, velars, etc.)
`dt`	→	3 (dentals)
`l`	→	4 (lateral)
`mn`	→	5 (nasals)
`r`	→	6 (rhotic)

Four step process:

`king`	`kyngge`
`k052`	`k05220`
`k052`	`k0520`
`k52`	`k52`

Soundex

One mechanism: Soundex

Translation table:

`aehiouwy`	→	0 (vowels)
`bpfv`	→	1 (labials)
`cgjkqsxz`	→	2 (misc: fricatives, velars, etc.)
`dt`	→	3 (dentals)
`l`	→	4 (lateral)
`mn`	→	5 (nasals)
`r`	→	6 (rhotic)

Four step process:

`knight`	`night`
`k50203`	`n0203`
`k50203`	`n0203`
`k523`	`n23`

Soundex

One mechanism: Soundex

Translation table:

`aehiouwy`	→	0 (vowels)
`bpfv`	→	1 (labials)
`cgjkqsxz`	→	2 (misc: fricatives, velars, etc.)
`dt`	→	3 (dentals)
`l`	→	4 (lateral)
`mn`	→	5 (nasals)
`r`	→	6 (rhotic)

Four step process:

`loan`	`loew`	`lough`	`lewicks`
`1005`	`1000`	`10020`	`1000222`
`105`	`10`	`1020`	`102`
`15`	`1`	`12`	`12`

Other Phonetic Methods

Better phonetic methods make use of the fact that some letters sounds alike in certain contexts, and different in other contexts

Editex uses the Edit Distance to compare strings based on a similar translation table to Soundex

lpadist uses a text–to–sound algorithm to represent tokens according to the International Phonetic Alphabet (but context matters a lot)

There are also worse variants, like Phonix.