Approximate Matching

COMP90049
Knowledge Technologies

Jeremy Nicholson, Justin Zobel, Karin Verspoor, Rao Kotagiri

Semester 1

THE UNIVERSITY OF MELBOURNE

Summary

Week 3:

- Approximate String Search and Matching
- Common Applications
- Methods:
  - Neighbourhood Search
  - Edit Distance
  - N-Gram Distance
  - [Phonetic methods]
- Evaluation
- [Genomics]

Exact String Search

Consider:

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

Exact String Search

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

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

Consider:

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

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?

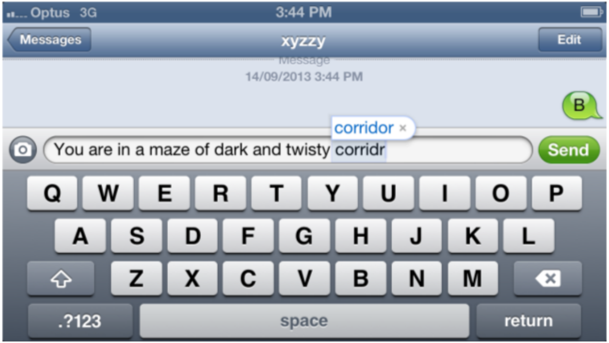
This is a Knowledge Technology!

Important problems

Two main applications for Approximate String Search:

- Spelling correction
- Computational Genomics

Spelling Correction



14/09/2013 3:44 PM

You are in a maze of dark and twisty corridr

corridor ×

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 or 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 or entries that are "correct"
- We can break our input into 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 or entries that are “correct”
- We can break our input into substrings that we wish to match, and compare each of them against the entries in the dictionary
- Aitem 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 or entries that are “correct”
- We can break our input into substrings that we wish to match, and compare each of them against the entries in the dictionary
- A item in the input which *doesn’t* appear in the dictionary is *misspelled*
- A item in the input which *does* appear in the dictionary might be correctly spelled *or* misspelled

Spelling Correction

Need the notion of a **dictionary**:

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

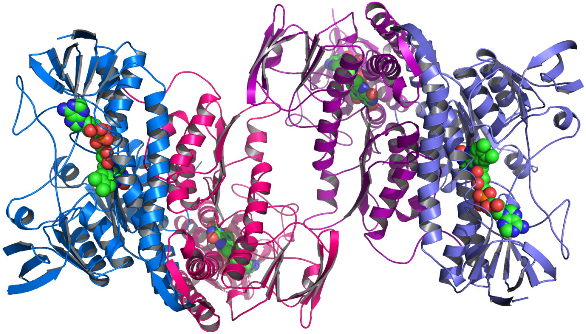
Therefore, the problem here:

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!

Other Problems of Interest

Computational Genomics (later, if we have time)



“Genomics”, Wikipedia, used under CC BY-SA 2.0
`https://creativecommons.org/licenses/by-sa/2.0/deed.en`

Other Problems of Interest

- Name matching, for example:

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, ...
```

Other Problems of Interest

- Computational Genomics (later, if we have time)
- Name matching
- Query repair
- Phonetic matching (later, if we have time)
- Data cleaning
- ...

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.
```

Insert `x` (and fold case)

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.
```

Delete `n`

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.
```

Replace `e` with `a`
(Sometimes **Substitute**)

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.
```

Transpose `e` and `o`
(Beyond the scope of this subject.)

Neighbourhood Search

For a given string *w* of interest:

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

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

Neighbourhood Search

For example:

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

Neighbourhood Search

For example:

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

Intended word: `other`

Requires 1 insertion (`o`) so intended word will be found using neighbourhood search (and some unintended words...)

Neighbourhood Search Efficiency

With a careful implementation, Neighbourhood search is suprisingly fast!

Neighbourhood Search Efficiency

Neighbourhood search is suprisingly fast!

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

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

Neighbourhood Search Efficiency

Neighbourhood search is suprisingly fast!

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

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

Neighbourhood Search Efficiency

Neighbourhood search is suprisingly fast!

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

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

Neighbourhood Search Efficiency

Neighbourhood search is suprisingly fast!

Consider: alphabet size is $|\Sigma|$, 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, and k is usually small

Neighbourhood Search Efficiency

Neighbourhood search is suprisingly fast!

Consider: alphabet size is $|\Sigma|$, 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, and k is usually small

For each neighbour, need a dictionary read (dict has D entries): Binary search yields $\mathcal{O}(|w|^k \log D)$ string comparisons

Neighbourhood Search Effectiveness

So, efficiency isn't our problem.

(`agrep` example)

Global Edit Distance

Alternative methods:

Scan through each dictionary entry looking for the “best” match

Global Edit Distance

Global Edit Distance:

Transform the string of interest into each dictionary entry, using the operations Insert, Delete, Replace, and Match (character)

Global Edit Distance

Global Edit Distance:

Transform the string of interest into each dictionary entry, using the operations Insert, Delete, Replace, and Match (character)

Each operation is associated with a score;
Best match is the dictionary entry with best aggregate **score**

Global Edit Distance

For example:

Item of interest: `crat`

Dictionary: `cart`, `arts`

Global Edit Distance

For example:

Item of interest: `crat`

Dictionary: `cart`, `arts`

`crat` → `cart`:
Match `c`, Delete `r`, Match `a`, Insert `r`, Match `t`

Global Edit Distance

For example:

Item of interest: `crat`

Dictionary: `cart`, `arts`

`crat` → `cart`:
Match `c`, Delete `r`, Match `a`, Insert `r`, Match `t`

`crat` → `arts`:
Replace `c` with `a`, Match `r`, Delete `a`, Match `t`, Insert s

Global Edit Distance

For example:

Item of interest: `crat`

Dictionary: `cart`, `arts`

Score: Match +1, Insert -1, Delete -1, Replace -1

`crat` → `cart`:
Match `c`, Delete `r`, Match `a`, Insert `r`, Match `t`

`crat` → `arts`:
Replace `c` with `a`, Match `r`, Delete `a`, Match `t`, Insert s

Global Edit Distance

For example:

Item of interest: `crat`

Dictionary: `cart`, `arts`

Score: Match +1, Insert -1, Delete -1, Replace -1

`crat` → `cart`:
Match `c` (+1), Delete `r` (-1), Match `a` (+1), Insert `r` (-1), Match `t` (+1) = +1

`crat` → `arts`:
Replace `c` with `a` (-1), Match `r` (+1), Delete `a` (-1), Match `t` (+1), Insert s (-1) = -1

`cart` is the better match

Global Edit Distance Parameter

Confusingly, Global Edit Distance isn’t a “distance”

Global Edit Distance Parameter

Confusingly, Global Edit Distance isn’t a “distance”

...But depends on parameter

Global Edit Distance Parameter

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

This is the Levenshtein Distance (which is a “distance”): it counts the number of edits required to transform one string into the other

Global Edit Distance Parameter

Hypothetically, any parameter is possible!

Global Edit Distance Parameter

Hypothetically, any parameter is possible!

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

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

Global Edit Distance 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 Parameter

Hypothetically, any parameter is possible!

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

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

`aba`: Which corresponds to best match?

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

Global Edit Distance Parameter

Hypothetically, any parameter is possible!

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

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

`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 Parameter

Often, “direction” doesn’t matter: Insert = Delete (“Indel”)

Global Edit Distance Parameter

Sometimes, score of Replace depends on which character is being replaced:

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

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 of $|f| + 1$ columns and $|t| + 1$ rows, we can solve using the Needleman–Wunsch algorithm:

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: As the algorithm progresses, the Aij will be assigned to be the optimal score for the alignment of the first j characters in f and the first i characters in t.

```
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
```

Note: *m*= cost of match; *r* = cost of replace; *i* = insertion cost; *d* = delete cost

`equal()` returns *m* (cost of match) if characters match, *r* (cost of replace) otherwise

Global Edit Distance in Action

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

Global Edit Distance in Action

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

	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)

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)

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)

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)

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)

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)

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)

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)

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)

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

```
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

```
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

```
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

```
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
```

→ Match score less than Insert/Delete/Replace

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

(Levenshtein Distance)

Local Edit Distance

Local Edit Distance is like Global Edit Distance, but we are searching for the best <u>substring</u> match

Local Edit Distance

Local Edit Distance is 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 word and a sentence

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)

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

Local Edit Distance in Action

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

	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)

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)

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)

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)

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)

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)

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)

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)

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)

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)

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

For strings f and t, Both algorithms above are $\mathcal{O}(|f||t|)$ in both space and time. (Space can be improved, but time (probably) cannot.)

Edit Distance Efficiency

When approximate matching, we have a constant string *f* which we want to compare to each string in the dictionary:

Edit Distance Efficiency

When approximate matching, we have a constant string *f* which we want to compare to each string *t* in the dictionary *D*:

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

Edit Distance Efficiency

When approximate matching, we have a constant string *f* which we want to compare to each string *t* in the dictionary *D*:

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

Edit Distance Efficiency

When approximate matching, we have a constant string *f* which we want to compare to each string *t* in the dictionary *D*:

Hence, integer comparisons are roughly the number of characters in the dictionary. Whether this is feasible depends on the size of the dictionary.

N-Gram Distance

N-Gram Distance has same goal as Edit Distance: compare two strings to determine “best” match

A true “distance”

N-Gram Distance

N-Gram Distance has same goal as Global Edit Distance, but much simpler

N-Gram Distance

(character) *n*-gram: substring of length *n*

N-Gram Distance

n-gram: substring of length *n*

2-grams of `crat`: `cr`, `ra`, `at`

N-Gram Distance

n-gram: substring of length *n*

2-grams of `crat`: `#c`, `cr`, `ra`, `at`, `t#` (sometimes)

N-Gram Distance

n-gram: substring of length *n*

3-grams of `crat`: `#cr`, `cra`, `rat`, `at#`

N-Gram Distance

n-gram: substring of length *n*

2-grams of `crat`: `#c`, `cr`, `ra`, `at`, `t#`
2-grams of `cart`: `#c`, `ca`, `ar`, `rt`, `t#`
2-grams of `arts`: `#a`, `ar`, `rt`, `ts`, `s#`

N-Gram Distance

n-gram: substring of length *n*

2-grams of `crat`: `#c`, `cr`, `ra`, `at`, `t#`
2-grams of `cart`: `#c`, `ca`, `ar`, `rt`, `t#`
2-grams of `arts`: `#a`, `ar`, `rt`, `ts`, `s#`

N-Gram Distance between *n*-grams of string s ($G_n(s)$) and t ($G_n(t)$):
$|G_n(s)| + |G_n(t)| - 2 \times |G_n(s) \cap G_n(t)|$

N-Gram Distance

n-gram: substring of length *n*

2-grams of `crat`: `#c`, `cr`, `ra`, `at`, `t#`
2-grams of `cart`: `#c`, `ca`, `ar`, `rt`, `t#`
2-grams of `arts`: `#a`, `ar`, `rt`, `ts`, `s#`

2-Gram Distance between `crat` and `cart`:

$$|G_2(\texttt{crat})| + |G_2(\texttt{cart})| - 2 \times |G_2(\texttt{crat}) \cap G_2(\texttt{cart})|$$
$$= 5 + 5 - 2 \times 2 = 6$$

N-Gram Distance

n-gram: substring of length *n*

2-grams of `crat`: `#c`, `cr`, `ra`, `at`, `t#`
2-grams of `cart`: `#c`, `ca`, `ar`, `rt`, `t#`
2-grams of `arts`: `#a`, `ar`, `rt`, `ts`, `s#`

2-Gram Distance between `crat` and `cart`:
$|G_2(\texttt{crat})| + |G_2(\texttt{cart})| - 2 \times |G_2(\texttt{crat}) \cap G_2(\texttt{cart})|$
$= 5 + 5 - 2 \times 2 = 6$
2-Gram Distance between `crat` and `arts`:
$|G_2(\texttt{crat})| + |G_2(\texttt{arts})| - 2 \times |G_2(\texttt{crat}) \cap G_2(\texttt{arts})|$
$= 5 + 5 - 2 \times 0 = 10$

N-Gram Distance

n-gram: substring of length *n*

2-grams of `crat`: `#c`, `cr`, `ra`, `at`, `t#`
2-grams of `cart`: `#c`, `ca`, `ar`, `rt`, `t#`
2-grams of `arts`: `#a`, `ar`, `rt`, `ts`, `s#`

2-Gram Distance between `crat` and `cart`:
$|G_2(\texttt{crat})| + |G_2(\texttt{cart})| - 2 \times |G_2(\texttt{crat}) \cap G_2(\texttt{cart})|$
$= 5 + 5 - 2 \times 2 = 6$ (better)
2-Gram Distance between `crat` and `arts`:
$|G_2(\texttt{crat})| + |G_2(\texttt{arts})| - 2 \times |G_2(\texttt{crat}) \cap G_2(\texttt{arts})|$
$= 5 + 5 - 2 \times 0 = 10$

N-Gram Distance Efficiency

Occasionally useful as a simpler variant of (Global) Edit Distance

N-Gram Distance Efficiency

Occasionally useful as a simpler variant of Edit Distance

More sensitive to long substring matches, less sensitive to relative ordering of strings (matches can be anywhere!)

N-Gram Distance Efficiency

Occasionally useful as a simpler variant of Edit Distance

More sensitive to long substring matches, less sensitive to relative ordering of strings (matches can be anywhere!)

Despite its simplicity, takes roughly the same time to compare entire dictionary

N-Gram Distance Efficiency

Occasionally useful as a simpler variant of Edit Distance

More sensitive to long substring matches, less sensitive to relative ordering of strings (matches can be anywhere!)

Despite its simplicity, takes roughly the same time to compare entire dictionary

Quite useless for very long strings and/or very small alphabets (Why?)

Orthography

In English (and some other languages), **orthography** (spelling) isn’t a good predictor of **phonetics** (sounds)

Orthography

In English (and some other languages), **orthography** (spelling) isn't a good predictor of **phonetics** (sounds)

Salient concern in speech–to–text systems, e.g.:
`Georgia Conal`

Orthography

In English (and some other languages), **orthography** (spelling) isn't a good predictor of **phonetics** (sounds)

Salient concern in speech–to–text systems, e.g.:

```
Georgia Conal
George O'Connell
```

Orthography

In English (and some other languages), **orthography** (spelling) isn’t a good predictor of **phonetics** (sounds)

Salient concern in speech–to–text systems, e.g.:

`You wreck a nice beach`

Orthography

In English (and some other languages), **orthography** (spelling) isn't a good predictor of **phonetics** (sounds)

Salient concern in speech–to–text systems, e.g.:

```
You wreck a nice beach
You recognize speech
```

Orthography

In English (and some other languages), **orthography** (spelling) isn't a good predictor of **phonetics** (sounds)

Salient concern in speech–to–text systems, e.g.:

`Lowe`

Orthography

In English (and some other languages), **orthography** (spelling) isn't a good predictor of **phonetics** (sounds)

Salient concern in speech–to–text systems, e.g.:

`Lowe`

`Lo`

Orthography

In English (and some other languages), **orthography** (spelling) isn't a good predictor of **phonetics** (sounds)

Salient concern in speech–to–text systems, e.g.:

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

Orthography

In English (and some other languages), **orthography** (spelling) isn't a good predictor of **phonetics** (sounds)

Also relevant in spelling correction (English can be very difficult to spell correctly!)

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

Ipadist 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.

Evaluating an Approximate Matching System

Evaluation: consider whether the system is effective at solving the user's problem

Evaluating an Approximate Matching System

Evaluation: consider whether the system is effective at solving the user's problem

In this case: for a misspelled word, does the system identify the correct word?

Evaluating an Approximate Matching System

To evaluate, we need:

- A number of cases of misspelled words

Evaluating an Approximate Matching System

To evaluate, we need:

- A number of cases of misspelled words
- The intended (correct) word for each case

Evaluating an Approximate Matching System

To evaluate, we need:

- A number of cases of misspelled words
- The intended (correct) word for each case
- An **evaluation metric**

Evaluation Metrics for Spelling Correction

We have some cases:

Evaluation Metrics for Spelling Correction

Misspelled Word	Correct Word
ther	other
corridr	corridor
cracheyt	crotchety
...	...

Evaluation Metrics for Spelling Correction

Misspelled Word	Correct Word	Predicted Word
ther	other	there
corridr	corridor	corridor
cracheyt	crotchety	cachet
...	...	...

Evaluation Metrics for Spelling Correction

Misspelled Word	Correct Word	Predicted Word	Right/Wrong?
ther	other	there	×
corridr	corridor	corridor	✓
cracheyt	crotchety	cachet	×
...	...	...	...

Evaluation Metrics for Spelling Correction

Misspelled Word	Correct Word	Predicted Word	Right/Wrong?
`ther`	`other`	`there`	×
`corridr`	`corridor`	`corridor`	✓
`cracheyt`	`crotchety`	`cachet`	×
...	...	...	...

Accuracy: fraction of correct responses ($\frac{1}{3}$)

Evaluation Metrics for Spelling Correction

Misspelled Word	Correct Word	Predicted Word	Right/Wrong?
`ther`	`other`	`there`	×
`corridr`	`corridor`	`corridor`	✓
`cracheyt`	`crotchety`	`cachet`	×
...	...	...	...

Accuracy: $\frac{\text{Number of correct predictions}}{\text{Total number of words}}$

Evaluation Metrics for Spelling Correction

More realistic situation:

Misspelled Word	Correct Word	Predicted Word
ther	other	there other their
corridr	corridor	corridor carrier
cracheyt	crotchety	???
...	...	...

Evaluation Metrics for Spelling Correction

More realistic situation:

Misspelled Word	Correct Word	Predicted Word	Right/Wrong?
`ther`	`other`	`there`	×
		`other`	✓
		`their`	×
`corridr`	`corridor`	`corridor`	✓
		`carrier`	×
`cracheyt`	`crotchety`	???	???
...	...	...	

Evaluation Metrics for Spelling Correction

Misspelled Word	Correct Word	Predicted Word	Right/Wrong?
`ther`	`other`	`there`	×
		`other`	✓
		`their`	×
`corridr`	`corridor`	`corridor`	✓
		`carrier`	×
`cracheyt`	`crotchety`	???	—
...	...	...	

Precision: fraction of correct responses among attempted responses ($\frac{2}{5}$)

Evaluation Metrics for Spelling Correction

Misspelled Word	Correct Word	Predicted Word	Right/Wrong?
ther	other	there	×
		other	✓
		their	×
corridr	corridor	corridor	✓
		carrier	×
cracheyt	crotchety	???	—
...	...	...	

Recall: proportion of words with a correct response (somewhere) $(\frac{2}{3})$

Comparing Systems

Typically, the value of the evaluation metric has little intrinsic meaning

Comparing Systems

Typically, the value of the evaluation metric has little intrinsic meaning

“This system gets 81% accuracy” — useful for users, or not?

Comparing Systems

The evaluation metric allows us to compare systems:

“The system based on the Global Edit Distance gets 81% accuracy, whereas the system based on the N-Gram Distance gets 84% accuracy”

Comparing Systems

The evaluation metric allows us to compare systems:

“The basic system gets 81% accuracy, but after making some changes, the accuracy becomes 74%”

Comparing Systems

Typically, comparison is more difficult:

“System A gets 45% precision and 80% recall;
System B gets 95% precision and 10% recall”

Comparing Systems

Typically, comparison is more difficult:

“System A gets 45% precision and 80% recall;
System B gets 95% precision and 10% recall”
— Which one should we use? (Also: why?)

Comparing Systems

The answer depends on the problem (and the user)!

Summary

- What is approximate string search?
- What are some common applications of approximate string search; why are they hard?
- What are some methods for finding an approximate match to a string? What do we need to generate them?
- How can we evaluate a typical approximate matching system?

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): 44353. 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): 707710.

Smith, Temple F. and Waterman, Michael S. (1981). “Identification of Common Molecular Subsequences”. Journal of Molecular Biology 147: 195197. doi:10.1016/0022-2836(81)90087-5

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.

Zobel, Justin and Dart, Philip (1996). “Phonetic String Matching: Lessons from Information Retrieval”. In Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR’96), pp. 166-172, New York, USA.

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.

Computational Genomics

Typical Genomics problem:

- Given a nucleotide/amino acid sequence (substring)
- Find whether the sequence occurs within a larger sequence (string)
- Possibly with “errors” (nucleotide/amino acid changes)

Computational Genomics

Typical Genomics problem:

- Given a substring, find whether the sequence occurs within a larger string, possibly with “errors”
- Almost the same as spelling correction, flipped around

Computational Genomics

Typical Genomics problem:

- Given a substring, find whether the sequence occurs within a larger string, possibly with “errors”
- Almost the same as spelling correction
- But **much** larger strings: a small genomics problem might involve comparing perhaps 1K character sequence against several 100K character sequences; alphabet is smaller

Suitability for Computational Genomics

Recall: we have a “short” ($\sim$1K character) nucleotide/amino acid sequence to compare against many long ($\sim$100K character) chromosomes/genes/proteins/etc.

Suitability for Computational Genomics

Recall: we have a “short” ($\sim$1K character) string to compare against many long ($\sim$100K character) strings

Suitability for Computational Genomics

Recall: we have a "short" (∼1K character) string to compare against many long (∼100K character) strings

For example, if some member of the population has 99% of the sequence of interest, they might be susceptible to some medical condition

Suitability for Computational Genomics

Recall: we have a “short” (∼1K character) string to compare against many long (∼100K character) strings

We’re allowed ∼10 errors; alphabet is ∼4 or ∼20 characters

Suitability for Computational Genomics

Neighbourhood search:

Roughly $4^{10} \times 1000^{10}$ possible neighbours.

Suitability for Computational Genomics

Neighbourhood search:

Roughly $4^{10} \times 1000^{10}$ possible neighbours.

... Forget it.

Suitability for Computational Genomics

Global Edit Distance:

One string is $\sim$1K characters, other is $\sim$100K characters.

Suitability for Computational Genomics

Global Edit Distance:

One string is ∼1K characters, other is ∼100K characters.

... Every string comparison involves ∼99K insertions.

→ Prefers shorter chromosomes (not intended behaviour)

Suitability for Computational Genomics

Local Edit Distance:

One string is ∼1K characters, other is ∼100K characters.

... Seems like the right idea.

Suitability for Computational Genomics

Local Edit Distance:

One string is ∼10K characters, other is ∼1G characters.

... Can’t fit table into memory.

Suitability for Computational Genomics

Local Edit Distance:

One string is ∼10K characters, other is ∼1G characters.

... Requires approximate solutions with heuristics, e.g. BLAST, FASTA

Suitability for Computational Genomics

N-Gram Distance:

With huge *n* (e.g. 80% of length of shorter string) can (almost) work!

Suitability for Computational Genomics

N-Gram Distance:

Surprisingly, can (almost) work!

Tends to prefer shorter chromosomes like Global Edit Distance

Suitability for Computational Genomics

N-Gram Distance:

But better methods for using *n*-gram information, e.g. de Bruijn graphs