Bioinformatische Methoden in der Historischen Linguistik *String Alignment*

Gerhard Jäger

8. Februar 2013
Forum Scientiarum

The Levenshtein Distance

- also known as *edit distance*
- defines the distance between two strings as the minimal number of *edit operations* to transform one string into the other
- edit operations:
  - deletion
  - insertion
  - replacemant
- example: grm. `mEnS` vs. Cimbrian `menEs`
  1. `mEnS` $\rightarrow$ `menS` (replace)
  2. `menS` $\rightarrow$ `menES` (insert)
  3. `menES` $\rightarrow$ `menEs` (insert)
- $d_L(\texttt{mEnS}, \texttt{menEs}) = 3$

The Levenshtein Distance

- alternative presentation: alignment

$$\begin{array}{ccccc} \mathtt{m} & \mathtt{E} & \mathtt{n} & - & \mathtt{S} \\ | & | & | & | & | \\ \mathtt{m} & \mathtt{e} & \mathtt{n} & \mathtt{E} & \mathtt{s} \end{array}$$

- distance for a particular alignment is the number of non-identities
- Levenshtein distance is the number of mismatches for the optimal alignment

Computing the Levenshtein Distance

- recursive definition:
  1. $d_L(\epsilon, \alpha) = d_L(\alpha, \epsilon) = l(\alpha)$
  2. $$d_L(\alpha x, \beta y) = \min \begin{cases} d_L(\alpha, \beta) + \delta(x, y) \\ d_L(\alpha x, \beta) + 1 \\ d_L(\alpha, \beta y) + 1 \end{cases}$$
- apparently requires exponentially growing number of comparisons $\Rightarrow$ computationally not feasible
- but:
  - if $l(\alpha) = n$ and $l(\beta) = m$, there are $n + 1$ substrings of α and $m + 1$ substrings of β
  - hence there are only $(n + 1)(m + 1)$ many different comparisons to be performed
  - computational complexity is polynomial (quadratic in $l(\alpha) + l(\beta)$)

Computing the Levenshtein Distance

- Dynamic Programming

	—	m	E	n	S
—	0	1	2	3	4
m	1				
e	2				
n	3				
E	4				
s	5				

Computing the Levenshtein Distance

- Dynamic Programming

	—	m	E	n	S
—	0	1	2	3	4
m	1				
e	2				
n	3				
E	4				
s	5				

Computing the Levenshtein Distance

- Dynamic Programming

	—	m	E	n	S
—	0	1	2	3	4
m	1				
e	2				
n	3				
E	4				
s	5				

Computing the Levenshtein Distance

- Dynamic Programming

	—	m	E	n	S
—	0	1	2	3	4
m	1	0			
e	2				
n	3				
E	4				
s	5				

Computing the Levenshtein Distance

- Dynamic Programming

	—	m	E	n	S
—	0	1	2	3	4
m	1	0	← ↖ ↑		
e	2				
n	3				
E	4				
s	5				

Computing the Levenshtein Distance

- Dynamic Programming

	—	m	E	n	S
—	0	1	2	3	4
m	1	0			
e	2				
n	3				
E	4				
s	5				

Computing the Levenshtein Distance

- Dynamic Programming

	—	m	E	n	S
—	0	1	2	3	4
m	1	0	1		
e	2				
n	3				
E	4				
s	5				

Computing the Levenshtein Distance

- Dynamic Programming

	–	m	E	n	S
–	0	1	2	3	4
m	1	0	1	2	
e	2				
n	3				
E	4				
s	5				

Computing the Levenshtein Distance

- Dynamic Programming

	—	m	E	n	S
—	0	1	2	3	4
m	1	0	1	2	3
e	2				
n	3				
E	4				
s	5				

Computing the Levenshtein Distance

- Dynamic Programming

	–	m	E	n	S
–	0	1	2	3	4
m	1	0	1	2	3
e	2	1			
n	3				
E	4				
s	5				

Computing the Levenshtein Distance

- Dynamic Programming

	–	m	E	n	S
–	0	1	2	3	4
m	1	0	1	2	3
e	2	1	1		
n	3				
E	4				
s	5				

Computing the Levenshtein Distance

- Dynamic Programming

	—	m	E	n	S
—	0	1	2	3	4
m	1	0	1	2	3
e	2	1	1	2	
n	3				
E	4				
s	5				

Computing the Levenshtein Distance

- Dynamic Programming

	—	m	E	n	S
—	0	1	2	3	4
m	1	0	1	2	3
e	2	1	1	2	3
n	3				
E	4				
s	5				

Computing the Levenshtein Distance

- Dynamic Programming

	—	m	E	n	S
—	0	1	2	3	4
m	1	0	1	2	3
e	2	1	1	2	3
n	3	2			
E	4				
s	5				

Computing the Levenshtein Distance

- Dynamic Programming

	—	m	E	n	S
—	0	1	2	3	4
m	1	0	1	2	3
e	2	1	1	2	3
n	3	2	2		
E	4				
s	5				

Computing the Levenshtein Distance

- Dynamic Programming

	—	m	E	n	S
—	0	1	2	3	4
m	1	0	1	2	3
e	2	1	1	2	3
n	3	2	2	1	
E	4				
s	5				

Computing the Levenshtein Distance

- Dynamic Programming

	—	m	E	n	S
—	0	1	2	3	4
m	1	0	1	2	3
e	2	1	1	2	3
n	3	2	2	1	2
E	4				
s	5				

Computing the Levenshtein Distance

- Dynamic Programming

	—	m	E	n	S
—	0	1	2	3	4
m	1	0	1	2	3
e	2	1	1	2	3
n	3	2	2	1	2
E	4	3			
s	5				

Computing the Levenshtein Distance

- Dynamic Programming

	—	m	E	n	S
—	0	1	2	3	4
m	1	0	1	2	3
e	2	1	1	2	3
n	3	2	2	1	2
E	4	3	2		
s	5				

Computing the Levenshtein Distance

- Dynamic Programming

	–	m	E	n	S
–	0	1	2	3	4
m	1	0	1	2	3
e	2	1	1	2	3
n	3	2	2	1	2
E	4	3	2	2	
s	5				

Computing the Levenshtein Distance

- Dynamic Programming

	—	m	E	n	S
—	0	1	2	3	4
m	1	0	1	2	3
e	2	1	1	2	3
n	3	2	2	1	2
E	4	3	2	2	2
s	5				

Computing the Levenshtein Distance

- Dynamic Programming

	—	m	E	n	S
—	0	1	2	3	4
m	1	0	1	2	3
e	2	1	1	2	3
n	3	2	2	1	2
E	4	3	2	2	2
s	5	4			

Computing the Levenshtein Distance

- Dynamic Programming

	—	m	E	n	S
—	0	1	2	3	4
m	1	0	1	2	3
e	2	1	1	2	3
n	3	2	2	1	2
E	4	3	2	2	2
s	5	4	3		

Computing the Levenshtein Distance

- Dynamic Programming

	—	m	E	n	S
—	0	1	2	3	4
m	1	0	1	2	3
e	2	1	1	2	3
n	3	2	2	1	2
E	4	3	2	2	2
s	5	4	3	3	

Computing the Levenshtein Distance

- Dynamic Programming

	—	m	E	n	S
—	0	1	2	3	4
m	1	0	1	2	3
e	2	1	1	2	3
n	3	2	2	1	2
E	4	3	2	2	2
s	5	4	3	3	3

Computing the Levenshtein Distance

- Dynamic Programming

	–	m	E	n	S
–	0	1	2	3	4
m	1	0	1	2	3
e	2	1	1	2	3
n	3	2	2	1	2
E	4	3	2	2	2
s	5	4	3	3	**3**

- memorizing in each step which of the three cells to the left and above gave rise to the current entry lets us recover the corresponing optimal alignment

Computing the Levenshtein Distance

- Dynamic Programming

	—	m	E	n	S
—	0	1	2	3	4
m	1	0	1	2	3
e	2	1	1	2	3
n	3	2	2	1	2
E	4	3	2	2	2
s	5	4	3	3	↖ 3

- memorizing in each step which of the three cells to the left and above gave rise to the current entry lets us recover the corresponing optimal alignment

Computing the Levenshtein Distance

- Dynamic Programming

	–	m	E	n	S
–	0	1	2	3	4
m	1	0	1	2	3
e	2	1	1	2	3
n	3	2	2	1	2
E	4	3	2	2 ↑	2
s	5	4	3	3	↖ 3

- memorizing in each step which of the three cells to the left and above gave rise to the current entry lets us recover the corresponing optimal alignment

Computing the Levenshtein Distance

- Dynamic Programming

	–	m	E	n	S
–	0	1	2	3	4
m	1	0	1	2	3
e	2	1	1	2	3
n	3	2	2	↖1	2
E	4	3	2	↑2	2
s	5	4	3	3	↖3

- memorizing in each step which of the three cells to the left and above gave rise to the current entry lets us recover the corresponing optimal alignment

Computing the Levenshtein Distance

- Dynamic Programming

	—	m	E	n	S
—	0	1	2	3	4
m	1	0	1	2	3
e	2	1	↖1	2	3
n	3	2	2	↖1	2
E	4	3	2	2↑	2
s	5	4	3	3	↖3

- memorizing in each step which of the three cells to the left and above gave rise to the current entry lets us recover the corresponing optimal alignment

Computing the Levenshtein Distance

- Dynamic Programming

	—	m	E	n	S
—	0	1	2	3	4
m	1	↖ 0	1	2	3
e	2	1	↖ 1	2	3
n	3	2	2	↖ 1	2
E	4	3	2	2 ↑	2
s	5	4	3	3	↖ 3

m	E	n	—	S
m	e	n	E	s

Computing the Levenshtein Distance

	—	m	E	n	S
—	0	1	2	3	4
m	1	↖0	1	2	3
e	2	1	↖1	2	3
n	3	2	2	↖1	2
E	4	3	2	2↑	2
s	5	4	3	3	↖3

m	E	n	—	S
m	e	n	E	s

	—	m	E	n	S
—	0	1	2	3	4
m	1	↖0	1	2	3
e	2	1	↖1	2	3
n	3	2	2	↖1	2
E	4	3	2	2	↖2
s	5	4	3	3	3↑

m	E	n	S	—
m	e	n	E	s

Normalization for length

- grm. `mEnS` (*Mensch*, 'person') and Hindi `manuSya` are (partially) cognate
- grm. `ze3n` (*sehen*, 'see') and Hindi `deg` are not cognate
- still

$$\begin{aligned} d_L(\texttt{mEnS}, \texttt{manuSya}) &= 4 \\ d_L(\texttt{ze3n}, \texttt{deg}) &= 3 \end{aligned}$$

- normalization: dividing Levenshtein distance by length of longer string:

$$\begin{aligned} d_{LD}(\texttt{mEnS}, \texttt{manuSya}) &= 4/7 \approx 0.57 \\ d_{LD}(\texttt{ze3n}, \texttt{deg}) &= 3/4 = 0.75 \end{aligned}$$

German — Swabian

'I':	0.5	iX	i-
'you':	0.0	du	du
'we':	0.67	vir	mia
'one':	0.5	ains	oi-s
'two':	0.25	cvai	cvoi
'person':	0.4	mEn-S	mEnZE
'fish':	0.0	fiS	fiS
'dog':	0.25	hunt	hund
'louse':	0.0	laus	laus
'tree':	0.5	baum	b-om
'leaf':	0.25	blat	blad
'skin':	0.0	haut	haut
'blood':	0.25	blut	blud
'bone':	0.33	knoX3n	knoX-E
'horn':	0.25	horn	hoan
'eye':	0.25	aug3	augE
'nose':	1.0	-naz3	ciNgE
'tooth':	0.25	chan	c-an
'knee':	0.25	kn-i	knui
'hand':	0.5	hant	hEnd
'breast':	0.4	brust	bXuSt
'liver':	0.4	leb3r	leb-a
'drink':	0.71	triNk3n	dXiN--g
'see':	0.75	ze3n	se--
'hear':	0.6	her3n	he--a
'die':	0.57	Sterb3n	StEab--
'come':	0.33	khom3n	khom--
'sun':	0.5	zon3	sonE
'star':	0.2	StErn	StEan
'water':	0.6	vas3r	va-za
'stone':	0.4	Stain	Stoi-
'fire':	0.25	foia	fuia

German — Swabian

```
'path':    1.0
pfat
-veg

'mountain':  0.5
bErk
bEag

'night':   0.33
nat
nad

'full':    0.0
fol
fol

'new':     0.0
noi
noi

'name':    0.5
nam3
nom-
```

German — English

'I':	1.0	iX	Ei
'you':	0.5	du	yu
'we':	0.67	vir	wi-
'one':	0.75	ains	w3n-
'two':	1.0	cvai	--tu
'fish':	0.0	fiS	fiS
'dog':	1.0	hunt	-dag
'louse':	0.0	laus	laus
'tree':	1.0	baum	-tri
'leaf':	0.75	blat	-lif
'blood':	0.5	blut	bl3d
'bone':	0.67	knoX3n	-bo--n
'horn':	0.0	horn	horn
'eye':	1.0	aug3	--Ei
'nose':	0.75	naz3	n-os
'tooth':	1.0	chan	-tu8
'tongue':	0.8	chuN3	-t3N-
'knee':	0.33	kni	-ni
'hand':	0.5	hant	hEnd
'breast':	0.15	brust	brest
'liver':	0.4	leb3r	liv3r
'drink':	0.57	triNk3n	drink--
'see':	1.0	ze3n	--si
'hear':	0.6	her3n	hir--
'die':	1.0	Sterb3n	----dEi
'come':	0.67	khom3n	k---3m
'sun':	0.75	zon3	s3n-
'star':	0.6	StErn	star-
'water':	0.4	vas3r	wat3r
'stone':	0.6	Stain	st-on
'fire':	0.5	foia	fEir
'path':	0.75	pfat	p-E8

German — Latin

'I':	1.0	-iX	ego
'you':	0.5	du	tu
'we':	1.0	vir	nos
'one':	0.75	ains	unus
'two':	1.0	cvai	-duo
'person':	0.86	---mEnS	persona
'fish':	0.83	---fiS	piskis
'dog':	0.8	hun-t	kanis
'louse':	0.78	-----laus	pedikulus
'tree':	1.0	-baum	arbor
'leaf':	0.8	-blat	folyu
'skin':	0.8	haut--	-kutis
'blood':	1.0	---blut	saNgwis
'bone':	0.83	knoX3n	--o--s
'horn':	0.4	horn-	kornu
'eye':	0.83	-au-g3	okulus
'nose':	0.6	na-z3	nasus
'tooth':	1.0	chan	dens
'tongue':	1.0	-chuN3	liNgwE
'knee':	0.75	-kni	genu
'hand':	0.6	han-t	manus
'breast':	0.83	--brust	pektus-
'liver':	0.6	leb3r	yekur
'drink':	0.86	triNk3n	-bibere
'see':	0.83	--ze3n	widere
'hear':	1.0	-her3n	audire
'die':	0.86	Sterb3n	-mor--i
'come':	1.0	khom3n	wenire
'sun':	0.75	zon3	so-l
'star':	0.8	StErn	stela
'water':	0.8	vas3r	-akwa
'stone':	0.8	Stain	lapis

German — Latin

```
'fire':   0.8
-foia
iNnis

'path':   1.0
pfat
viya

'mountain': 1.0
bErk
mons

'night':  0.75
n-at
noks

'full':   1.0
---fol
plenus

'new':    0.6
no--i
nowus

'name':   0.6
nam-3
nomen
```

Evaluation: cognates

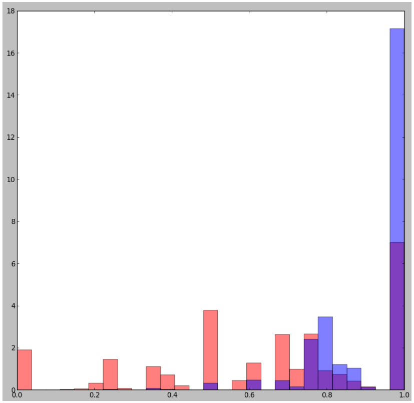
```
0.0
['fiS' 'German_ST' 'fiS' 'English_ST']
0.2
['leb3r' 'German_ST' 'lev3r' 'Dutch_List']
0.2
['leb3r' 'German_ST' 'lev3r' 'Afrikaans']
0.25
['hunt' 'German_ST' 'hont' 'Afrikaans']
0.25
['hunt' 'German_ST' 'hun' 'Kashmiri']
0.25
['hunt' 'German_ST' 'hont' 'Dutch_List']
0.25
['hunt' 'German_ST' 'hun7' 'Danish']
0.4
['leb3r' 'German_ST' 'liv3r' 'English_ST']
0.43
['triNk3n' 'German_ST' 'driNk' 'Afrikaans']
0.5
['leb3r' 'German_ST' 'levEr3' 'Flemish']
0.5
['hant' 'German_ST' 'hEnd' 'Swedish_Up']
0.5
['hant' 'German_ST' 'hEnd' 'English_ST']
0.5
['foia' 'German_ST' 'fir' 'Flemish']
0.5
['blut' 'German_ST' 'bl3d' 'English_ST']
0.5
['hunt' 'German_ST' 'ont' 'Flemish']
```

```
1.0
['aug3' 'German_ST' 'oko' 'BULGARIAN_P']
1.0
['aug3' 'German_ST' 'voka' 'BYELORUSSIAN_P']
1.0
['aug3' 'German_ST' 'oko' 'MACEDONIAN_P']
1.0
['aug3' 'German_ST' 'mati' 'Greek_Mod']
1.0
['aug3' 'German_ST' 'oko' 'Polish']
1.0
['aug3' 'German_ST' 'voka' 'Byelorussian']
1.0
['aug3' 'German_ST' 'oko' 'Czech_E']
1.0
['aug3' 'German_ST' 'yakh' 'Gypsy_Gk']
1.0
['hunt' 'German_ST' 'kau' 'Portuguese_ST']
1.0
['aug3' 'German_ST' 'okyo' 'Italian']
1.0
['aug3' 'German_ST' 'oky' 'Rumanian_List']
1.0
['aug3' 'German_ST' '3y' 'French']
1.0
['hunt' 'German_ST' 'sp3i' 'Afghan']
1.0
['aug3' 'German_ST' 'oko' 'Bulgarian']
1.0
['aug3' 'German_ST' 'oho' 'Spanish']
```

Evaluation: non-cognates

```
0.33
['uL' 'Catalan' 'suL' 'Irish_A' 'EYE']
0.33
['sag' 'Persian_List' 'dag' 'English_ST' 'DOG']
0.33
['sag' 'Tadzik' 'dag' 'English_ST' 'DOG']
0.33
['mau' 'Portuguese_ST' 'Lau' 'Welsh_C' 'HAND']
0.33
['ble' 'Faroese' 'le' 'Singhalese' 'BLOOD']
0.4
['foia' 'German_ST' 'fotya' 'Greek_Mod' 'FIRE']
0.4
['Zuvis' 'Lithuanian_ST' 'vis' 'Dutch_List' 'FISH']
0.4
['lamo' 'Nepali_List' 'largo' 'Spanish' 'LONG']
0.5
['zivs' 'Latvian' 'fis' 'Afrikaans' 'FISH']
0.5
['kan' 'Bengali' 'skuarn' 'Breton_ST' 'EAR']
```

Evaluation



- data from overlap Dyen-Kruskal data base/ASJP
- blue: non-cognates
- red: cognates
- mean normalized distance:
  - cognates: 0.648
  - non-cognates: 0.915

Problems

- binary distinction: match vs. non-match
- frequently genuin sound correspondences in cognates are missed:

c	v	a	i		n	a	**z**	3		-	-	-	**f**	i	S
-	-	**t**	u		n	-	o	**s**		**p**	i	s	k	i	s

- corresponding sounds count as mismatches even if they are aligend correctly

h	a	n	t		h	a	n	t
h	E	n	d		m	a	n	o

- substantial amount of chance similarities

Background: probability theory

- Given two sequences: How likely is it that they are aligned?
- More general question: Given some data, and two competing hypotheses, how likely is it that the first hypothesis is correct?

Bayesian Inference!!!

- given:
  - data: d
  - hypotheses: h_1, h_0
  - model: $P(d|h_1), P(d|h_0)$
- wanted:

$$P(h_1|d) : P(h_0|d)$$

Bayesian inference

- Bayes Theorem:

$$P(h|d) = \frac{P(d|h)P(h)}{\sum_{h'} P(d|h')P(h')}$$

- ergo:

$$\begin{aligned}
P(h_1|d) : P(h_0|d) &= P(d|h_1)P(h_1) : P(d|h_0)P(h_0) \\
P(h_1|d) : P(h_0|d) &= \frac{P(d|h_1)}{P(d|h_0)} \frac{P(h_1)}{P(h_0)} \\
\log(P(h_1|d) : P(h_0|d)) &= \log \frac{P(d|h_1)}{P(d|h_0)} + \log \frac{P(h_1)}{P(h_0)}
\end{aligned}$$

Bayesian inference

- suppose we have many independent data: $\vec{d} = d_1, \ldots, d_n$

$$
\begin{aligned}
P(\vec{d}|h) &= \prod_{i=1}^{n} P(d_i|h) \\
\log P(\vec{d}|h) &= \sum_{i=1}^{n} \log P(d_i|h) \\
\log \frac{P(\vec{d}|h_1)}{P(\vec{d}|h_0)} &= \sum_{i=1}^{n} \log \frac{P(d_i|h_1)}{P(d_i|h_0)} \\
\log(P(h_1|\vec{d}) : P(h_0|\vec{d})) &= \sum_{i=1}^{n} \log \frac{P(d_i|h_1)}{P(d_i|h_0)} + \log \frac{P(h_1)}{P(h_0)}
\end{aligned}
$$

Bayesian inference

- mein argument against using Bayes' rule: the **prior probabilities** $P(h_1), P(h_0)$ are not known
- there are various heuristics, but no generally accepted way to obtain them
- if n is large though, $\log {}^{P(h_1)}/_{P(h_0)}$ doesn't matter very much:[1]

$$\log(P(h_1|\vec{d}) : P(h_0|\vec{d})) \quad \approx \quad \sum_{i=1}^{n} \log \frac{P(d_i|h_1)}{P(d_i|h_0)} = \log(P(\vec{d}|h_1) : P(\vec{d}|h_0))$$

- the quantity $\log(P(\vec{d}|h_1) : P(\vec{d}|h_0))$ is called **log-odds**

[1] Also, if we choose an *uninformative prior* with $P(h_1) = P(h_0)$, we have $\log {}^{P(h_1)}/_{P(h_0)} = 0$ anyway.

Log-odds

- log-odds can take any real value
- a positive value indicates evidence for h_1 and a negative value evidence for h_0
- the higher the absolute value, the stronger is the evidence

Weighted alignment

- suppose our data are two aligned sequences $\vec{x}$, $\vec{y}$
- for the time being, we assume there are no gaps in the alignment
  - h_1: they developed from a common ancestor via substitions
  - h_0: they are unrelated
- additional assumptions (rough approximation in biology, pretty much off the mark in linguistics): substitions in different positions occur independently

The null model

- if $\vec{x}$ and $\vec{y}$ are unrelated, their joint probability equals the product of their individual probabilities
- as a start (quite wrong both in biology and in linguistics): let us assume the strings have no "grammar"; each position is independent from all other positions
- then

$$\begin{aligned} P(\vec{x},\vec{y}|h_0) &= P(\vec{x}|h_0)P(\vec{y}|h_0) \\ &= \prod_i P(x_i|h_0)P(y_i|h_0) \\ \log P(\vec{x},\vec{y}|h_0) &= \sum_i \log(P(x_i|h_0) + \log P(y_i|h_0)) \end{aligned}$$

The null model

- suppose $\vec{x}$ and $\vec{y}$ are generated by the same process (reasonable for DNA and protein comparison, false for cross-linguistic word comparison)
- then $P(x_i|h), P(y_i|h)$ are simply the probabilities of occurrence
- q_a: probability that symbol a occurs in a sequence

$$\log P(\vec{x}, \vec{y}|h_0) = \sum_i \log q_{x_i} + \sum_j \log q_{y_j}$$

- q can be estimated from relative frequencies

The alignment model

- suppose $\vec{x}$ and $\vec{y}$ evolved from a common ancestor via independent substitution mutations
- independence between positions:

$$P(\vec{x}, \vec{y} | h_1) = \prod_i P(x_i, y_i | h_2)$$

- $p_{a,b}$: probability that a position in the latest common ancestor of x and y evolved into an a in sequence $\vec{x}$ and into a b in sequence $\vec{y}$

$$\begin{aligned} P(\vec{x}, \vec{y} | h_1) &= \prod_i p_{x_i, y_i} \\ \log P(\vec{x}, \vec{y} | h_1) &= \sum_i \log p_{x_i, y_i} \end{aligned}$$

The log-odds score

- taking things together, we have

$$\log(P(\vec{x},\vec{y}|h_1) : P(\vec{x},\vec{y}|h_0)) = \sum_i \log \frac{p_{x_i,y_i}}{q_{x_i} q_{y_i}}$$

- $\log \frac{p_{ab}}{q_a q_b}$: **score** of the alignment of *a* with *b*
- assembled in a **substitution matrix**

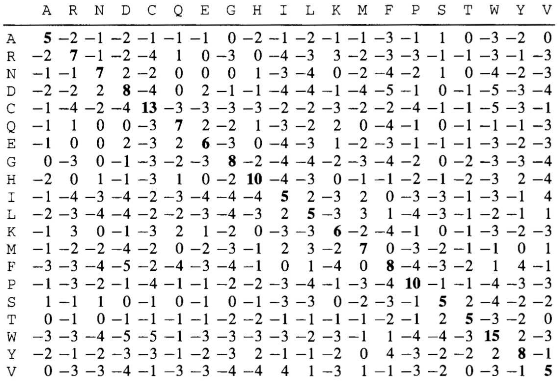
Substitution matrices

- in bioinformatics, several commonly used substitution matrices for nucleotids and proteins
- based on explicit models of evolution and careful empirical testing
- for nucleotids:

$$\begin{array}{ccccc} & A & G & T & C \\ A & 2 & -5 & -7 & -7 \\ G & -5 & 2 & -7 & -7 \\ T & -7 & -7 & 2 & -5 \\ C & -7 & -7 & -5 & 2 \end{array}$$

Substitution matrices

- for proteins: different matrices for different evolutionary distances
- for instance: BLOSUM50



	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	**5**	−2	−1	−2	−1	−1	−1	0	−2	−1	−2	−1	−1	−3	−1	1	0	−3	−2	0
R	−2	**7**	−1	−2	−4	1	0	−3	0	−4	−3	3	−2	−3	−3	−1	−1	−3	−1	−3
N	−1	−1	**7**	2	−2	0	0	0	1	−3	−4	0	−2	−4	−2	1	0	−4	−2	−3
D	−2	−2	2	**8**	−4	0	2	−1	−1	−4	−4	−1	−4	−5	−1	0	−1	−5	−3	−4
C	−1	−4	−2	−4	**13**	−3	−3	−3	−3	−2	−2	−3	−2	−2	−4	−1	−1	−5	−3	−1
Q	−1	1	0	0	−3	**7**	2	−2	1	−3	−2	2	0	−4	−1	0	−1	−1	−1	−3
E	−1	0	0	2	−3	2	**6**	−3	0	−4	−3	1	−2	−3	−1	−1	−1	−3	−2	−3
G	0	−3	0	−1	−3	−2	−3	**8**	−2	−4	−4	−2	−3	−4	−2	0	−2	−3	−3	−4
H	−2	0	1	−1	−3	1	0	−2	**10**	−4	−3	0	−1	−1	−2	−1	−2	−3	2	−4
I	−1	−4	−3	−4	−2	−3	−4	−4	−4	**5**	2	−3	2	0	−3	−3	−1	−3	−1	4
L	−2	−3	−4	−4	−2	−2	−3	−4	−3	2	**5**	−3	3	1	−4	−3	−1	−2	−1	1
K	−1	3	0	−1	−3	2	1	−2	0	−3	−3	**6**	−2	−4	−1	0	−1	−3	−2	−3
M	−1	−2	−2	−4	−2	0	−2	−3	−1	2	3	−2	**7**	0	−3	−2	−1	−1	0	1
F	−3	−3	−4	−5	−2	−4	−3	−4	−1	0	1	−4	0	**8**	−4	−3	−2	1	4	−1
P	−1	−3	−2	−1	−4	−1	−1	−2	−2	−3	−4	−1	−3	−4	**10**	−1	−1	−4	−3	−3
S	1	−1	1	0	−1	0	−1	0	−1	−3	−3	0	−2	−3	−1	**5**	2	−4	−2	−2
T	0	−1	0	−1	−1	−1	−1	−2	−2	−1	−1	−1	−1	−2	−1	2	**5**	−3	−2	0
W	−3	−3	−4	−5	−5	−1	−3	−3	−3	−3	−2	−3	−1	1	−4	−4	−3	**15**	2	−3
Y	−2	−1	−2	−3	−3	−1	−2	−3	2	−1	−1	−2	0	4	−3	−2	−2	2	**8**	−1
V	0	−3	−3	−4	−1	−3	−3	−4	−4	4	1	−3	1	−1	−3	−2	0	−3	−1	**5**

Substitution matrix for the ASJP data

1. identify large sample of pairs of closely related languages (using expert information or heuristics based on aggregated Levenshtein distance)

```
An.NORTHERN_PHILIPPINES.CENTRAL_BONTOC           An.SOUTHERN_PHILIPPINES.KAGAYANEN
An.MESO-PHILIPPINE.NORTHERN_SORSOGON             An.NORTHERN_PHILIPPINES.LIMOS_KALINGA

WF.WESTERN_FLY.IAMEGA                            An.MESO-PHILIPPINE.CANIPAAN_PALAWAN
WF.WESTERN_FLY.GAMAEWE                           An.NORTHWEST_MALAYO-POLYNESIAN.LAHANAN

Pan.PANOAN.KASHIBO_BAJO_AGUAYTIA                 NC.BANTOID.LIFONGA
Pan.PANOAN.KASHIBO_SAN_ALEJANDRO                 NC.BANTOID.BOMBOMA_2

AA.EASTERN_CUSHITIC.KAMBAATA_2                   IE.INDIC.WAD_PAGGA
AA.EASTERN_CUSHITIC.HADIYYA_2                    IE.INDIC.TALAGANG_HINDKO

ST.BAI.QILIQIAO_BAI_2                            NC.BANTOID.LINGALA
ST.BAI.YUNLONG_BAI                               NC.BANTOID.LIFONGA

An.SULAWESI.MANDAR                               An.CENTRAL_MALAYO-POLYNESIAN.BALILEDO
An.OCEANIC.RAGA                                  An.CENTRAL_MALAYO-POLYNESIAN.PALUE

An.SULAWESI.TANETE                               AuA.MUNDA.HO
An.SAMA-BAJAW.BOEPINANG_BAJAU                    AuA.MUNDA.KORKU

UA.AZTECAN.NAHUATL_HUEYAPAN_TETELA_DEL_VOLCAN    MGe.GE-KAINGANG.KAYAPO
UA.AZTECAN.NAHUATL_CUENTEPEC_TEMIXCO             MGe.GE-KAINGANG.APINAYE
```

Substitution matrix for the ASJP data

2. pick a concept and a pair of related languages at random
   - languages: `Pen.MAIDUAN.MAIDU_KONKAU`, `Pen.MAIDUAN.NE_MAIDU`
   - concept: *one*
3. find corresponding words from the two languages:
   - `nisam`, `niSem`
4. do Levenshtein alignment

n	i	s	a	m
n	i	S	e	m

5. for each sound pair, count number of correspondences
   - nn: 1; ii: 1; sS; 1; ae: 1; mm: 1

Substitution matrix for the ASJP data

- steps 2-5 are repeated 100,000 times

```
klem   S3--v   ligini   kulox   Naltir---i   ...
klom   S37on   ji---p   Gulox   Naltirtiri   ...
```

a	a	56,047	⋮	⋮	⋮
i	i	33,955	4	8	2
u	u	23,731	4	a	2
n	n	21,363	G	t	2
o	o	19,619	i	!	2
m	m	18,263	G	y	2
t	t	16,975	d	!	2
k	k	16,773	s	G	2
e	e	12,745	Z	5	2
r	r	11,601	G	s	2
l	l	11,377	X	z	2
b	b	8,965	!	k	2
s	s	8,245	q	8	2
d	d	6,829	a	!	2
p	p	6,681	a	!	2
w	w	6,613	!	y	2
N	N	6,275	!	E	2
h	h	5,331	j	G	2
y	y	5,321	G	i	2
3	3	5,255	E	!	2
⋮	⋮	⋮	v	S	2

Substitution matrix for the ASJP data

6. determine relative frequency of occurrence of each sound within the entire database

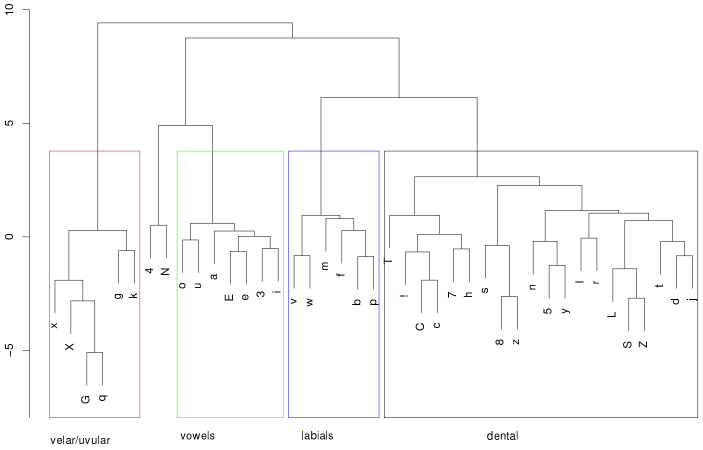
a	0.1479	E	0.0134
i	0.0969	7	0.0124
u	0.0696	C	0.0073
o	0.0626	S	0.0064
n	0.0614	x	0.0062
e	0.0478	c	0.0056
k	0.0478	f	0.0052
m	0.0465	5	0.0049
t	0.0449	v	0.0045
r	0.0346	q	0.0041
l	0.0331	z	0.0035
b	0.0248	j	0.0035
s	0.0243	T	0.0029
w	0.0232	L	0.0027
3	0.0228	X	0.0022
y	0.0222	8	0.0014
d	0.0214	Z	0.0011
h	0.0213	!	0.0009
p	0.0202	4	0.0002
N	0.0201	G	0.0001
g	0.0178		

Substitution matrix for the ASJP data

7. estimate p_{ab} as relative frequency of co-occurrence of a with b, q_a, q_b as individual relative frequencies, and determine substitution scores $\log_2 \frac{p_{ab}}{q_a q_b}$

G	G	11.2348	**Z**	**j**	**4.9386**		o	q	-3.2842
!	!	10.0202	d	d	4.9263		C	a	-3.2893
4	4	9.1480	g	g	4.8958		j	o	-3.2914
8	8	8.0650	b	b	4.8906		a	m	-3.2915
Z	Z	7.9575	s	s	4.8277		E	v	-3.3035
X	X	7.9375	**4**	**5**	**4.7508**		!	w	-3.3079
L	L	7.6276	E	E	4.7143		!	u	-3.3087
z	z	7.2624	w	w	4.6512		5	q	-3.3116
q	q	7.2542	h	h	4.5819		T	o	-3.3158
f	f	6.9117	**G**	**x**	**4.5573**		!	k	-3.3526
v	v	6.8418	**Z**	**z**	**4.4943**		e	z	-3.3763
5	5	6.7731	y	y	4.4637	. . .	!	s	-3.3788
j	j	6.7587	l	l	4.4037		f	q	-3.3942
T	T	6.6580	**!**	**G**	**4.3760**		N	S	-3.3954
S	S	6.6054	3	3	4.3692		!	b	-3.4077
c	c	6.5989	r	r	4.3061		L	b	-3.4558
C	C	6.2439	**X**	**q**	**4.1200**		T	u	-3.4690
4	**G**	**6.1943**	m	m	4.1087		4	i	-3.5529
x	x	6.1210	t	t	4.1021		5	a	-3.8294
G	**X**	**5.3342**	**G**	**Z**	**4.0429**		C	N	-3.8451
G	**q**	**5.3017**	k	k	3.9046		!	t	-4.2625
7	7	5.2111	**X**	**x**	**3.8116**		!	e	-4.3534
p	p	5.0693	**T**	**Z**	**3.7380**		!	i	-4.3712
N	N	4.9821	**8**	**G**	**3.6993**		!	a	-4.9817

Evaluation



10
5
0
−5
x
X
G
q
g
k
4
N
o
u
a
E
e
3
i
v
w
m
f
b
p
T
!
C
c
7
h
s
8
z
n
5
y
l
r
L
S
Z
t
d
j
velar/uvular
vowels
labials
dental

Gap penalties

- gaps in an alignment correspond either to an insertion or a deletion
- simplified assumption: insertions and deletions are equally likely at all positions; symbols are inserted according to their general frequency of occurrence
- Suppose an item x_i is aligned to a gap. Let α be the probability that an insertion occured since the latest common ancestor, and β the probability of a deletion

$$\begin{aligned} P(x_i, -|h_1) &= \alpha q_{x_i} + \beta q_{x_i} \\ P(x_i, -|h_0) &= q_{x_i} \\ \log(P(x_i, -|h_1) : P(x_i, -|h_0)) &= \log(\alpha + \beta) \\ &= -d \end{aligned}$$

- i.e., there is a constant term for each gap
- as $\alpha + \beta < 1$, this term is negative, i.e. there a constant **penalty** for each gap

Affine gap penalties

- deletions/insertions frequently apply to entire blocks of symbols (both in biology and linguistics)
- probability of a gap of length n are higher than the product of probabilities of n individual gaps
- penalty e for **extending** a gap is lower than penalty d for **opening** a gap
- g: length of a gap

$$\gamma(g) = -d - (g-1)e$$

- no principled way to derive the values of d and e; have to be fixed via trial and error
- $d = 2.5$ and $e = 1.6$ work quite well for the ASJP data

Weighted alignment

- so far, we assumed that the alignment between $\vec{x}$ and $\vec{y}$ is known
- to assess strength of evidence for h_1 given $\vec{x}, \vec{y}$, we need to consider all alignments between $\vec{x}$ and $\vec{y}$
- enumeration is infeasible, because the number of alignments between two sequences of length n is

$$\binom{2n}{n} = \frac{(2n)!}{(n!)^2} \approx \frac{2^{2^n}}{\sqrt{\pi n}}$$

- computation is nonetheless possible using *Pair Hidden Markov Models* (stay tuned for the next lecture!)
- simpler task: find the most likely alignment and determine its log-odds!

The Needleman-Wunsch algorithm

- almost identical to Levenshtein algorithm, except:
    - matches/mismatches are counted not as 1 and 0, but as log-odds scores of the corresponding symbol pair
    - insertions/deletions are counted as gap penalties
    - by convention, the similarity rather than the distance is counted, i.e. we try to find the alignment that maximizes the score
- let $\vec{x}$ have length n, $\vec{y}$ lenth m, s_{ab} be the log-odds score of a and b, and d/e the gap penalties

The Needleman-Wunsch algorithm

$$
\begin{aligned}
F(0,0) &= 0\\
G(0,0) &= 0\\
\forall i: &\quad 0 < i \leq n\\
F(i,0) &= F(i-1,0) + G(i-1,0)e + (1 - G(i-1,0))d\\
G(i,0) &= 1\\
\forall j: &\quad 0 < j \leq m:\\
F(0,j) &= F(0,j-1) + G(0,j-1)e + (1 - G(0,j-1))d\\
G(0,j) &= 1\\
\forall i,j: &\quad 0 < i \leq n, 0 < j \leq m\\
F(i,j) &= \max \left\{ \begin{array}{l} F(i-1,j) + G(i-1,j)e + (1 - G(i-1,j))d \\ F(i,j-1) + G(i,j-1) + (1 - G(i,j-1))d \\ F(i-1,j-1) + s_{x_i y_j} \end{array} \right.\\
G(i,j) &= 0 \text{ if } \arg\max \left\{ \begin{array}{l} F(i-1,j) + G(i-1,j)e + (1 - G(i-1,j))d \\ F(i,j-1) + G(i,j-1)e + (1 - G(i,j-1))d \\ F(i-1,j-1) + s_{x_i y_j} \end{array} \right\} = 3\\
&\quad 1 \text{ else}
\end{aligned}
$$

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5				
e	−4.1				
n	−5.7				
E	−7.3				
s	−8.9				

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5	$\nwarrow\uparrow\leftarrow$			
e	−4.1				
n	−5.7				
E	−7.3				
s	−8.9				

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5				
e	−4.1				
n	−5.7				
E	−7.3				
s	−8.9				

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5	4.13			
e	−4.1				
n	−5.7				
E	−7.3				
s	−8.9				

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5	4.13			
e	−4.1				
n	−5.7				
E	−7.3				
s	−8.9				

Computing the weighted alignment score

- Dynamic Programming

$$\begin{tikzcd}
 & \texttt{–} & \texttt{m} & \texttt{E} & \texttt{n} & \texttt{S} \\
\texttt{–} & 0 & -2.5 & -4.1 & -5.7 & -7.3 \\
\texttt{m} & -2.5 & 4.13 & {} \arrow[ul] \arrow[u] \arrow[l, red] & & \\
\texttt{e} & -4.1 & & & & \\
\texttt{n} & -5.7 & & & & \\
\texttt{E} & -7.3 & & & & \\
\texttt{s} & -8.9 & & & &
\end{tikzcd}$$

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5	4.13	1.53		
e	−4.1				
n	−5.7				
E	−7.3				
s	−8.9				

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5	4.13	1.53	0.03	
e	−4.1				
n	−5.7				
E	−7.3				
s	−8.9				

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5	4.13	1.53	0.03	−1.47
e	−4.1				
n	−5.7				
E	−7.3				
s	−8.9				

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5	4.13	1.53	0.03	−1.47
e	−4.1	1.53			
n	−5.7				
E	−7.3				
s	−8.9				

Computing the weighted alignment score

- Dynamic Programming

	−	m	E	n	S
−	0	−2.5	−4.1	−5.7	−7.3
m	−2.5	4.13	1.53	0.03	−1.47
e	−4.1	1.53	5.65		
n	−5.7				
E	−7.3				
s	−8.9				

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5	4.13	1.53	0.03	−1.47
e	−4.1	1.53	5.65	3.05	
n	−5.7				
E	−7.3				
s	−8.9				

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5	4.13	1.53	0.03	−1.47
e	−4.1	1.53	5.65	3.05	1.55
n	−5.7				
E	−7.3				
s	−8.9				

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5	4.13	1.53	0.03	−1.47
e	−4.1	1.53	5.65	3.05	1.55
n	−5.7	0.03			
E	−7.3				
s	−8.9				

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5	4.13	1.53	0.03	−1.47
e	−4.1	1.53	5.65	3.05	1.55
n	−5.7	0.03	3.05		
E	−7.3				
s	−8.9				

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5	4.13	1.53	0.03	−1.47
e	−4.1	1.53	5.65	3.05	1.55
n	−5.7	0.03	3.05	9.2	
E	−7.3				
s	−8.9				

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5	4.13	1.53	0.03	−1.47
e	−4.1	1.53	5.65	3.05	1.55
n	−5.7	0.03	3.05	9.2	6.6
E	−7.3				
s	−8.9				

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5	4.13	1.53	0.03	−1.47
e	−4.1	1.53	5.65	3.05	1.55
n	−5.7	0.03	3.05	9.2	6.6
E	−7.3	−1.47			
s	−8.9				

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5	4.13	1.53	0.03	−1.47
e	−4.1	1.53	5.65	3.05	1.55
n	−5.7	0.03	3.05	9.2	6.6
E	−7.3	−1.47	4.75		
s	−8.9				

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5	4.13	1.53	0.03	−1.47
e	−4.1	1.53	5.65	3.05	1.55
n	−5.7	0.03	3.05	9.2	6.6
E	−7.3	−1.47	4.75	6.6	
s	−8.9				

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5	4.13	1.53	0.03	−1.47
e	−4.1	1.53	5.65	3.05	1.55
n	−5.7	0.03	3.05	9.2	6.6
E	−7.3	−1.47	4.75	6.6	7.62
s	−8.9				

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5	4.13	1.53	0.03	−1.47
e	−4.1	1.53	5.65	3.05	1.55
n	−5.7	0.03	3.05	9.2	6.6
E	−7.3	−1.47	4.75	6.6	7.62
s	−8.9	−2.97			

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5	4.13	1.53	0.03	−1.47
e	−4.1	1.53	5.65	3.05	1.55
n	−5.7	0.03	3.05	9.2	6.6
E	−7.3	−1.47	4.75	6.6	7.62
s	−8.9	−2.97	2.15		

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5	4.13	1.53	0.03	−1.47
e	−4.1	1.53	5.65	3.05	1.55
n	−5.7	0.03	3.05	9.2	6.6
E	−7.3	−1.47	4.75	6.6	7.62
s	−8.9	−2.97	2.15	5.1	

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5	4.13	1.53	0.03	−1.47
e	−4.1	1.53	5.65	3.05	1.55
n	−5.7	0.03	3.05	9.2	6.6
E	−7.3	−1.47	4.75	6.6	7.62
s	−8.9	−2.97	2.15	5.1	8.84

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5	4.13	1.53	0.03	−1.47
e	−4.1	1.53	5.65	3.05	1.55
n	−5.7	0.03	3.05	9.2	6.6
E	−7.3	−1.47	4.75	6.6	7.62
s	−8.9	−2.97	2.15	5.1	8.84

- memorizing in each step which of the three cells to the left and above gave rise to the current entry lets us recover the corresponing optimal alignment

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5	4.13	1.53	0.03	−1.47
e	−4.1	1.53	5.65	3.05	1.55
n	−5.7	0.03	3.05	9.2	6.6
E	−7.3	−1.47	4.75	6.6 ↑	7.62
s	−8.9	−2.97	2.15	5.1	↖ 8.84

- memorizing in each step which of the three cells to the left and above gave rise to the current entry lets us recover the corresponing optimal alignment

Computing the weighted alignment score

- Dynamic Programming

$$
\begin{array}{c|ccccc}
 & - & \texttt{m} & \texttt{E} & \texttt{n} & \texttt{S} \\
\hline
- & 0 & -2.5 & -4.1 & -5.7 & -7.3 \\
\texttt{m} & -2.5 & 4.13 & 1.53 & 0.03 & -1.47 \\
\texttt{e} & -4.1 & 1.53 & 5.65 & 3.05 & 1.55 \\
\texttt{n} & -5.7 & 0.03 & 3.05 & \nwarrow 9.2 \uparrow & 6.6 \\
\texttt{E} & -7.3 & -1.47 & 4.75 & 6.6 & 7.62 \\
\texttt{s} & -8.9 & -2.97 & 2.15 & 5.1 & \nwarrow 8.84
\end{array}
$$

- memorizing in each step which of the three cells to the left and above gave rise to the current entry lets us recover the corresponing optimal alignment

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5	4.13	1.53	0.03	−1.47
e	−4.1	1.53	↖ 5.65	3.05	1.55
n	−5.7	0.03	3.05	↖ 9.2	6.6
E	−7.3	−1.47	4.75	6.6 ↑	7.62
s	−8.9	−2.97	2.15	5.1	↖ 8.84

- memorizing in each step which of the three cells to the left and above gave rise to the current entry lets us recover the corresponing optimal alignment

Computing the weighted alignment score

- Dynamic Programming

	–	m	E	n	S
–	0	−2.5	−4.1	−5.7	−7.3
m	−2.5	↖ 4.13	1.53	0.03	−1.47
e	−4.1	1.53	↖ 5.65	3.05	1.55
n	−5.7	0.03	3.05	↖ 9.2	6.6
E	−7.3	−1.47	4.75	↑ 6.6	7.62
s	−8.9	−2.97	2.15	5.1	↖ 8.84

- memorizing in each step which of the three cells to the left and above gave rise to the current entry lets us recover the corresponing optimal alignment

Evaluation

- scores:
  - s_{dt}: 0.27
  - s_{aE}: 0.19
  - s_{hm}: -1.76
  - s_{to}: -2.78
- $d_{NW}(hant, hEnd) = 8.59$
- $d_{NW}(hant, mano) = 1.40$

Evaluation

left: Levenshtein alignment; right: Needleman-Wunsch alignment

```
-iX          iX-            -blat        b-lat          han-t        han-t
ego          ego            folyu        folyu          manus        manus

du           du             haut--       haut--         --brust      b--rust
tu           tu             -kutis       k-utis         pektus-      pektus-

vir          vir            ---blut      ---blut        leb3r        leb3r
nos          nos            saNgwis      saNgwis        yekur        yekur

ains         ain-s          knoX3n       knoX3n         triNk3n      triNk3n-
unus         -unus          --o--s       --os--         -bibere      -bi-bere

cvai         cvai           horn-        horn-          --ze3n       --ze3n
-duo         duo-           kornu        kornu          widere       widere

---mEnS      mEnS---        -au-g3       a-ug3-         -her3n       --her3n
persona      persona        okulus       okulus         audire       audire-

---fiS       fiS---         na-z3        naz3-          Sterb3n      Sterb3n
piskis       piskis         nasus        nasus          -mor--i      -mor-i-

hun-t        hun-t          chan         chan-          khom3n       khom3n---
kanis        kanis          dens         d-ens          wenire       w---enire

-----laus    ------laus     -chuN3       chuN--3        zon3         zon3
pedikulus    pedikul-us     liNgwE       -liNgwE        so-l         sol-

-baum        --baum         -kni         k-ni           StErn        StErn
arbor        arb-or         genu         genu           stela        stela
```

Evaluation

```
vas3r         --vas3r
-akwa         akwa---

Stain         Sta-in
lapis         -lapis

-foia         fo-ia
iNnis         iNnis

pfat          p-fat
viya          viya-

bErk          bErk
mons          mons

n-at          na-t
noks          noks

---fol        fol----
plenus        p-lenus

no--i         no-i-
nowus         nowus

nam-3         nam3-
nomen         nomen
```

German — Swabian

'I':	0.3	iX	i
'you':	8.26	du	du
'we':	-1.09	vir	mia
'one':	4.63	ains	ois
'two':	16.0	cvai	cvoi
'person':	12.61	mEnS	mEnZE
'fish':	16.35	fiS	fiS
'dog':	11.76	hunt	hund
'louse':	15.01	laus	laus
'tree':	6.57	baum	bom
'leaf':	11.92	blat	blad
'skin':	14.42	haut	haut
'blood':	12.88	blut	blud
'bone':	16.88	knoX3n	knoXE
'horn':	8.75	horn	hoan
'tooth':	10.03	chan	can
'tongue':	9.8	chuN3	cuN
'knee':	7.77	kni	knui
'hand':	8.6	hant	hEnd
'breast':	14.81	brust	bXuSt
'liver':	10.01	leb3r	leba
'drink':	4.99	triNk3n	dXiNg
'see':	0.63	ze3n	se
'hear':	2.74	her3n	hea
'die':	10.16	Sterb3n	StEab
'come':	11.84	khom3n	khom
'sun':	8.79	zon3	sonE
'star':	16.16	StErn	StEan
'water':	7.8	vas3r	vaza
'stone':	10.36	Stain	Stoi
'fire':	12.43	foia	fuia
'path':	-2.57	pfat	veg

German — English

'I':	-2.3	iX	Ei
'you':	2.34	du	yu
'we':	2.21	vir	wi
'one':	-2.3	ains	w3n
'two':	-5.25	cvai	tu
'fish':	16.35	fiS	fiS
'dog':	-7.46	hunt	dag
'louse':	15.01	laus	laus
'tree':	-7.83	baum	tri
'leaf':	-0.47	blat	lif
'blood':	9.46	blut	bl3d
'bone':	-1.36	knoX3n	bon
'horn':	15.73	horn	horn
'eye':	-4.1	aug3	Ei
'nose':	1.63	naz3	nos
'tooth':	-6.23	chan	tu8
'tongue':	-0.63	chuN3	t3N
'knee':	3.86	kni	ni
'hand':	8.6	hant	hEnd
'breast':	16.93	brust	brest
'liver':	14.65	leb3r	liv3r
'drink':	7.48	triNk3n	drink
'see':	-3.04	ze3n	si
'hear':	4.61	her3n	hir
'die':	-7.7	Sterb3n	dEi
'come':	1.22	khom3n	k3m
'sun':	1.95	zon3	s3n
'star':	8.2	StErn	star
'water':	12.06	vas3r	wat3r
'stone':	6.75	Stain	ston
'fire':	6.79	foia	fEir
'path':	4.02	pfat	pE8

German — Latin

'I':	-3.87	iX	ego
'you':	3.62	du	tu
'we':	-5.06	vir	nos
'one':	2.39	ains	unus
'two':	-5.51	cvai	duo
'person':	-4.66	mEnS	persona
'fish':	0.29	fiS	piskis
'dog':	-2.27	hunt	kanis
'louse':	-0.08	laus	pedikulus
'tree':	-3.85	baum	arbor
'leaf':	-3.57	blat	folyu
'skin':	-0.25	haut	kutis
'blood':	-9.18	blut	saNgwis
'bone':	-5.72	knoX3n	os
'horn':	7.55	horn	kornu
'eye':	-3.87	aug3	okulus
'nose':	4.49	naz3	nasus
'tooth':	-2.78	chan	dens
'tongue':	-3.4	chuN3	liNgwE
'knee':	0.8	kni	genu
'hand':	0.73	hant	manus
'breast':	1.39	brust	pektus
'liver':	5.37	leb3r	yekur
'drink':	-9.22	triNk3n	bibere
'see':	-4.15	ze3n	widere
'hear':	-4.24	her3n	audire
'die':	-6.12	Sterb3n	mori
'come':	-9.25	khom3n	wenire
'sun':	0.97	zon3	sol
'star':	5.72	StErn	stela
'water':	-5.4	vas3r	akwa
'stone':	-3.26	Stain	lapis

Multiple sequence alignment

- Needleman-Wunsch and pair-HMMs only do pairwise alignment
- desirable: aligning all sequences of a taxon into one matrix
  - necessary for character-based phylogenetic inference
  - improves the quality of the alignment

Multiple sequence alignment

- example: 'one'
  - PIE: oinos
  - Bosian: yedan
  - Kashubian: yEdEn
  - optimal pairwise alignments:

o	i	n	o	s	o	i	n	o	s	y	e	d	a	n
y	e	d	a	n	y	E	d	E	n	y	E	d	E	n

- optimal multiple alignment (maximizing sum of pairwise similarities per column):

y	E	d	E	n	-	-
-	o	-	i	n	o	s
y	e	d	a	n	-	-

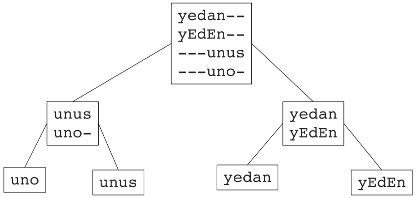
- alignment of all 'n's is etymologically correct

Multiple sequence alignment

- in principle, the Needleman-Wunsch algorithm can be generalized to aligning k sequences
- however, aligning k sequences of length n has complexity $\mathcal{O}(n^{k^2}) \Rightarrow$ computationally intractable
- two strategies
  - heuristic search
  - progressive alignment

Progressive sequence alignment

- start with a **guide tree** (using some heuristics like pairwise alignment + Neighbor Joining)
- working bottom-up, at each internal node, do pairwise alignment of the block alignments at the daugher node
- complexity is $\mathcal{O}(n^2k^3) \Rightarrow$ computationally feasible


yedan--
yEdEn--
---unus
---uno-
unus
uno-
yedan
yEdEn
uno
unus
yedan
yEdEn