جامعة المنوفية

MENOUFIA UNIVERSITY
FACULTY OF COMPUTERS AND INFORMATION

Fourth Year (Second Semester)
CS Dept., (CS 436)

Natural Language Processing NLP

Lecture six

Dr. Hamdy M. Mousa

PROBABILISTIC MODELS OF PRONUNCIATION AND SPELLING

SPELLING

- We discuss the problem of:
  - Detecting and correcting spelling errors
  - Modeling pronunciation variation for automatic speech recognition
  - Text-to-speech systems.
- On the surface, the problems of finding spelling errors in text and modeling the variable pronunciation of words in spoken language don't seem to have much in common.

SPELLING

- Similarly, given the sequence of letters corresponding to a misspelled word, we need to produce an ordered list of possible correct words.

- **Example:** the sequence **acress** might be a misspelling of **actress**, or of **cress**, or of **acres**.
- We transduce from the 'surface' form **acress** to the various possible 'lexical' forms, assigning each with a probability; we then select the most probable correct word.

Spelling Correction

- **Spelling correction** breaks the field down into three problems:
    1. **non-word error detection:** detecting spelling errors which result in non-words (like graffe for giraffe).
    2. **isolated-word error correction:** correcting spelling errors which result in non-words, for example correcting graffe to giraffe.
    3. **context-dependent error detection and correction:** Using the context to help detect and correct spelling errors even if they accidentally result in an actual word. (e.g. **there for three**, or **dessert for desert**, or **piece for peace**).

SPELLING ERROR PATTERNS

- Single-error misspellings:
  - insertion: mistyping the as ther
  - deletion: mistyping the as th
  - substitution: mistyping the as thw
  - transposition: mistyping the as hte

DETECTING NON-WORD ERRORS

- Because of the need to represent productive inflection (the -s and ed suffixes) and derivation.
  - Dictionaries for spelling error detection usually include ***models of morphology***,
    - just as the dictionaries for text-to-speech
  - Modern spelling error detectors use more linguistically-motivated morphological representations

Human Word Prediction

- Some of us have the ability to predict future words in an utterance
- **Related to:**
  - Domain knowledge
  - Syntactic knowledge
  - Lexical knowledge
- A useful part of the knowledge is needed to allow Word Prediction (guessing the next word).
- Word Prediction can be captured using simple statistical techniques

Predict?

- Why would you want to assign a probability to a sentence or …

- Why would you want to predict the next word…????

Applications

- Language models applications Example:
  - Speech recognition
  - Handwriting recognition
  - Spelling correction
  - Machine translation systems
  - Optical character recognizers
  - Text summarization
  - ………..

Handwriting Recognition

- Assume:

I have a gub.

- NLP to the rescue ….
  - gub is not a word
  - gun, gum, Gus, and gull are words
  - but gun has a higher probability in the context of a bank

For Spell Checkers

- Collect list of commonly substituted words
  - **piece/peace**, whether/weather, **die / dye** their/there ...

- Example:
  "On Tuesday, the whether …"
  "On Tuesday, the weather …"
- Example:
  - Everything We See Will Dye.
  - Everything We See Will Die.

Probability Distribution

- Statistical NLP aims to do statistical inference for the field of NL
- *Statistical inference* consists of taking some data (generated in accordance with some unknown *probability distribution*) and then making some inference about this distribution.
- In order to predict the next word, we need a *model* of the language.
- Probability theory helps us finding such model

Language Models

- It is an abstract representation of a (natural) language phenomenon. (an approximation to real language)
- help a speech recognizer figure out how likely a word sequence is, independent of the acoustics.
- A lot of candidates can be eliminated and it is possible to give other words higher probabilities.

Language Model

This lets the recognizer make the right guess when two different sentences

Example:

- **It’s fun to recognize speech?**
- **It’s fun to wreck a nice beach?**

>> (Sound the same)

Probability Theory

- How likely it is that an A Event (something) will happen
- Sample space Ω is listing of all possible outcome of an experiment
- Event A is a subset of Ω
- Probability function (or distribution)

$$P : \Omega \rightarrow [0,1]$$

Prior Probability

- **Prior (unconditional) probability**:
  - the probability before we consider any additional knowledge

$$P(A)$$

Conditional Probability

- Sometimes we have partial knowledge about the outcome of an experiment
- **Conditional Probability**
  - Suppose we know that event B is true
  - The probability that event A is true given the knowledge about B is expressed by

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

$$P(A \cap B) = P(A|B) \,.\, P(B)$$

Bayesian Inference

$$P(x|y) = \frac{P(y \cap x)}{P(y)} = \frac{P(y|x).P(x)}{P(y)}$$

- find out all possible words
- the word that is most likely given the observation

  → argmax P(w|O)

$$\begin{aligned} \operatorname{argmax}_{\text{Input}} P(\text{Input} \,|\, \text{Output}) &= \operatorname{argmax}_{\text{Input}} \frac{P(\text{Input})P(\text{Output} \,|\, \text{Input})}{P(\text{Output})} \\ &= \operatorname{argmax}_{\text{Input}} P(\text{Input})P(\text{Output} \,|\, \text{Input}) \end{aligned}$$

Bayesian Inference

$$\mathrm{argmax}_{\mathrm{Input}}\ P(\mathrm{Input} \mid \mathrm{Output}) = \mathrm{argmax}_{\mathrm{Input}}\ \underbrace{P(\mathrm{Input})}_{\text{Prior (language Model)}}\ \underbrace{P(\mathrm{Output} \mid \mathrm{Input})}_{\text{Likelihood (Domain Model)}}$$

	P(Input)	P(Output|Input)
Machine Translation	Language model	Translation model
OCR	Language model	Model of OCR errors
Spellchecking	Language model	Model of spelling errors
POS-Tagging	Language Model	Tag-Word Model
Speech Recognition	Language model	Acoustic model

Automatic Speech Recognition (ASR)

$$\arg\max_{wordsequence} P(wordsequence \mid acoustics) =$$

$$\arg\max_{wordsequence} \frac{P(acoustics \mid wordsequence) \times P(wordsequence)}{P(acoustics)}$$

$$\arg\max_{wordsequence} P(acoustics \mid wordsequence) \times P(wordsequence)$$

Machine Translation

$$\underset{wordsequence}{\arg\max}\, P(wordsequence \mid acoustics) =$$

$$\underset{wordsequence}{\arg\max}\, \frac{P(acoustics \mid wordsequence) \acute{}\ \ P(wordsequence)}{P(acoustics)}$$

$$\underset{wordsequence}{\arg\max}\, P(acoustics \mid wordsequence) \acute{}\ \ P(wordsequence)$$

$$\underset{wordsequence}{\arg\max}\, P(english \mid french) =$$

$$\underset{wordsequence}{\arg\max}\, \frac{P(french \mid english) \acute{}\ \ P(english)}{P(french)}$$

$$\underset{wordsequence}{\arg\max}\, P(french \mid english) \acute{}\ \ P(english)$$

Language Model

- **Most common: n-gram models**
- **data driven: given $n_1,n_2,n_3,n_4,\ldots n_z$**

  unigram: P(word) = freq(word) / N

  bigram: $P(word_i|word_{i-1})$

  $= freq(‘word_{i-1}\ word_i’)/freq(word_{i-1})$

  trigram…

  - **The higher *n*, the more context is captured**
  - **The higher *n*, the less statistical evidence we find for each context: sparse data problem**

Probabilistic Spelling Correction

- **Kernighan et al (1990): misspelt word differs from correct word in 1 substitution, insertion, transposition or deletion.**

error	**Correction**	**Correct Letter**	**Error Letter**	**Position**	**Type**
acress	actress	t	-	2	deletion
acress	cress	-	a	0	insertion
acress	caress	ca	ac	0	transposition
acress	access	c	r	2	substitution
…					

Probabilistic Spelling Correction

$$\text{correction} = \underset{c \in C}{\operatorname{argmax}}\, P(t|c).P(c)$$

- t : typo
- C : list of correct words

- P(c): prior: language model (unigram)
- P(t | c): Model of misspellings

Probabilistic Spelling Correction

- **Kernighan: 44×10^6 word AP newswire corpus**

PRIOR:

c	Freq(c)	P(c)
actress	1343	.0000315
cress	0	.000000014
caress	4	.0000001
access	2280	.000058

Probabilistic Spelling Correction

- Model of misspellings: P(t | c)
- Proper P(t | c) cannot be computed, but can be estimated
- Use corpus of errors to construct confusion matrix of 26x26 for each type of mistake
  - del[x,y]: count how many times xy was typed as x
  - ins[x,y]: count how many times x was types as xy
  - sub[x,y]: count how many times x was typed as y
  - trans[x,y]: how many times xy was types as yx

Probabilistic Spelling Correction

Correction	P(c)	P(t|c)	p(t|c) p(c)
actress	.0000315	.000117	3.69×10^{-9}
cress	.000000014	.00000144	2.02×10^{-14}
caress	.0000001	.0000164	1.64×10^{-13}
access	.000058	.000000209	1.21×10^{-11}

- **acress is rewritten as ‘actress’**
- **use more intelligent prior to improve results in context**

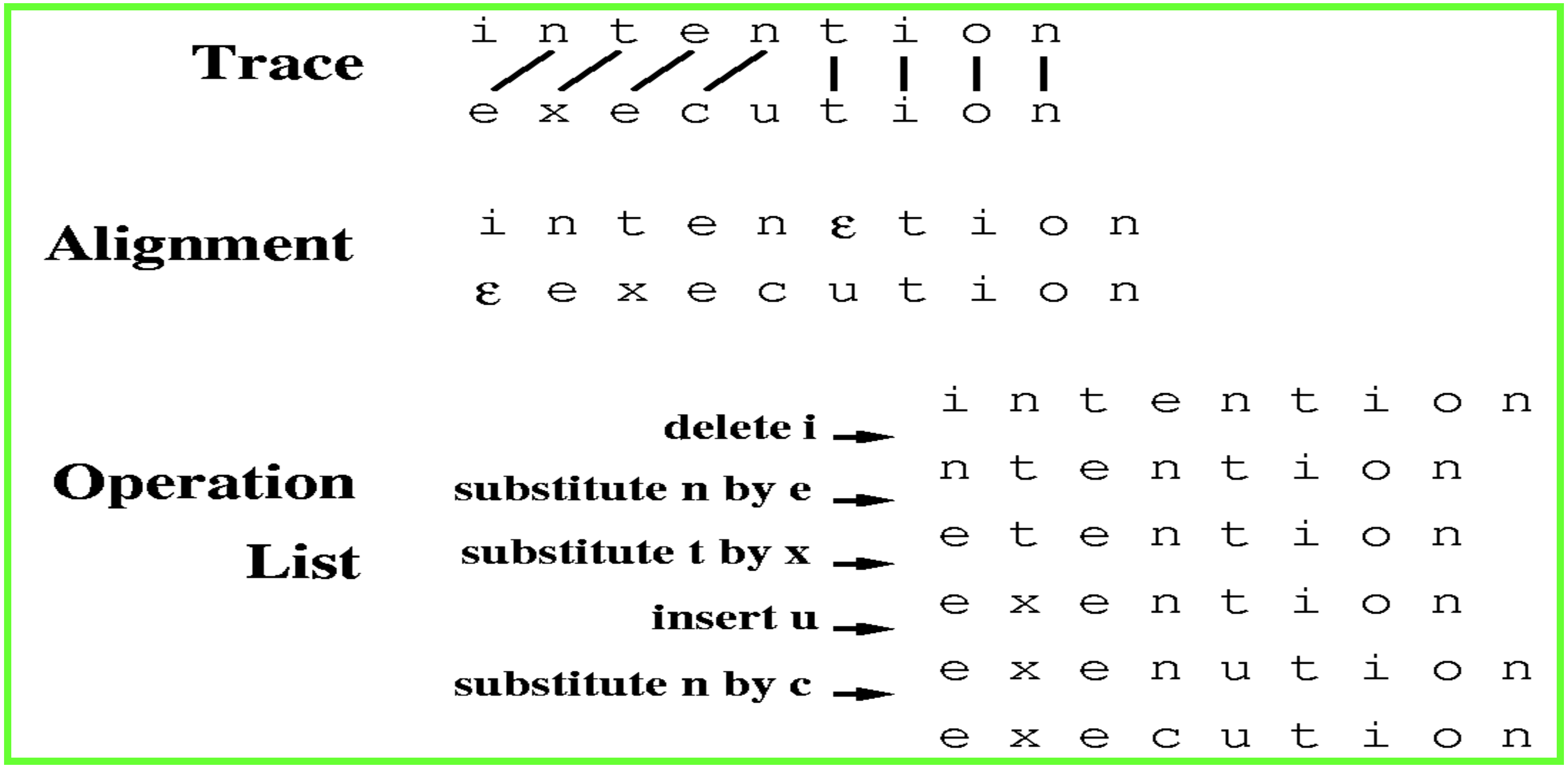
PROBABILISTIC MODELS OF PRONUNCIATION AND SPELLING

Minimum Edit Distance

- Kernighan et al. (1990) relied on the simplifying assumption that each word had only a single spelling error.
- In the case of multiple errors?
- the Bayesian algorithm implemented with confusion matrices, was able to rank candidate corrections.
- The **string distance** is some metric of how alike two strings are to each other.
- The **minimum edit distance** between two strings is the minimum number of editing operations (insertion, deletion, substitution) needed to transform one string into another.

Minimum Edit Distance

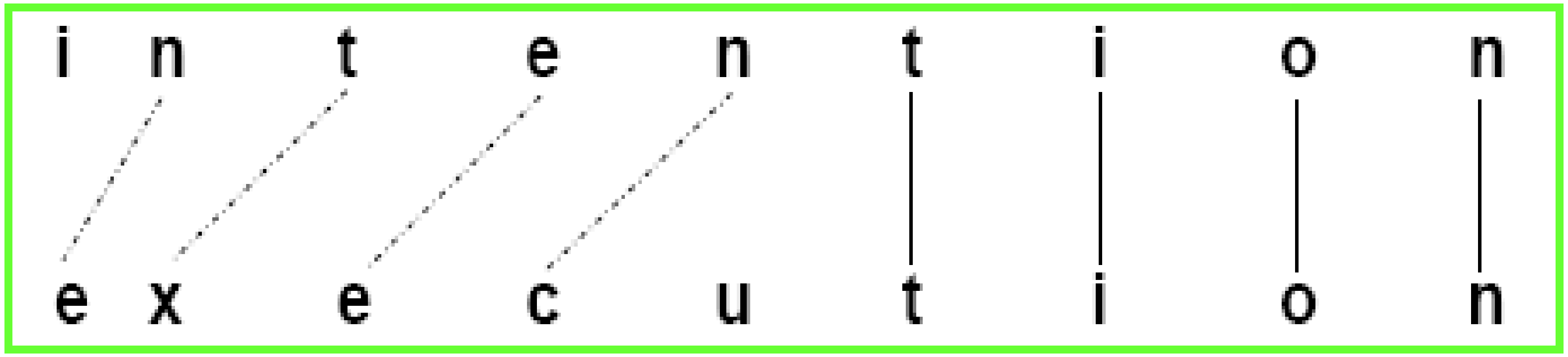
- For example the gap between intention and execution is 5 operations, which can be represented in three ways; as a trace, an alignment, or a operation list



- **Thus the Levenshtein distance between intention and execution is 5**

Minimum Edit Distance

- Different way to compute P(t | c)



- Difference of 5 letters: 5 operations

intention►ntention►etention►exention►exenution►execution

- Calculate number of deletions, insertion, substitutions
- Levenshtein Distance: equal weight to all
- maximum probability alignment (confusion matrix)

Minimum Edit Distance

- **Implemented using Dynamic Programming**
- **combine solutions to subproblems to overall solution execution vs intention**
- **look at each deletion, substitution, … on local level**
- **maintain results in chart**

- **Edit distance matrix:**
  - **Target = columns, source = rows**

Minimum Edit Distance

- Searching for a path (sequence of edits) from the start string to the final string:
  - **Initial state:** the word we’re transforming
  - **Operators:** insert, delete, substitute
  - **Goal state:** the word we’re trying to get to
  - **Path cost:** what we want to minimize: the number of edits

The Minimum Edit Distance Algorithm

```
function MIN-EDIT-DISTANCE(target, source) returns min-distance

  $n \leftarrow$ LENGTH(target)
  $m \leftarrow$ LENGTH(source)
  Create a distance matrix $distance[n+1,m+1]$
  $distance[0,0] \leftarrow 0$
  for each column $i$ from 0 to $n$ do
    for each row $j$ from 0 to $m$ do
      $distance[i,j] \leftarrow$ MIN( $distance[i-1,j] + ins\text{-}cost(target_j)$,
                                      $distance[i-1,j-1] + subst\text{-}cost(source_j, \text{target}_i)$,
                                      $distance[i,j-1] + ins\text{-}cost(source_j)$)
```

Trace to Minimum Edit Distance

Base conditions:

D(i,0) = i   D(0, j) = j

Termination:

D(N,M) is distance

Recurrence Relation*:*

For each i = 1…M

For each j = 1…N

$$D(i,j) = \min \begin{cases} D(i-1, j) + 1 \\ D(i, j-1) + 1 \\ D(i-1,j-1) + 2; & \text{if } X(i) \neq Y(j) \\ D(i-1,j-1) + 0; & \text{if } X(i) = Y(j) \end{cases}$$

Deletion (LEFT)

Insertion(DOWN)

Substitution(DIAG.)

Computation of Minimum Edit Distance

Insertion(DOWN)

Source (j)

n	9	10	11	10	11	12	11	10	9	**8**
o	8	9	10	9	10	11	10	9	**8**	9
i	7	8	9	8	9	10	9	**8**	9	10
t	6	7	8	7	8	9	**8**	9	10	11
n	5	6	7	6	7	**8**	9	10	11	12
e	4	5	6	**5**	**6**	7	8	9	10	11
t	3	4	**5**	6	7	8	9	10	11	12
n	2	3	**4**	5	6	7	8	8	10	11
i	1	**2**	3	4	5	6	7	8	9	10
#	**0**	1	2	3	4	5	6	7	8	9
	#	e	x	e	c	u	t	i	o	n

Substitution(DIAG.)

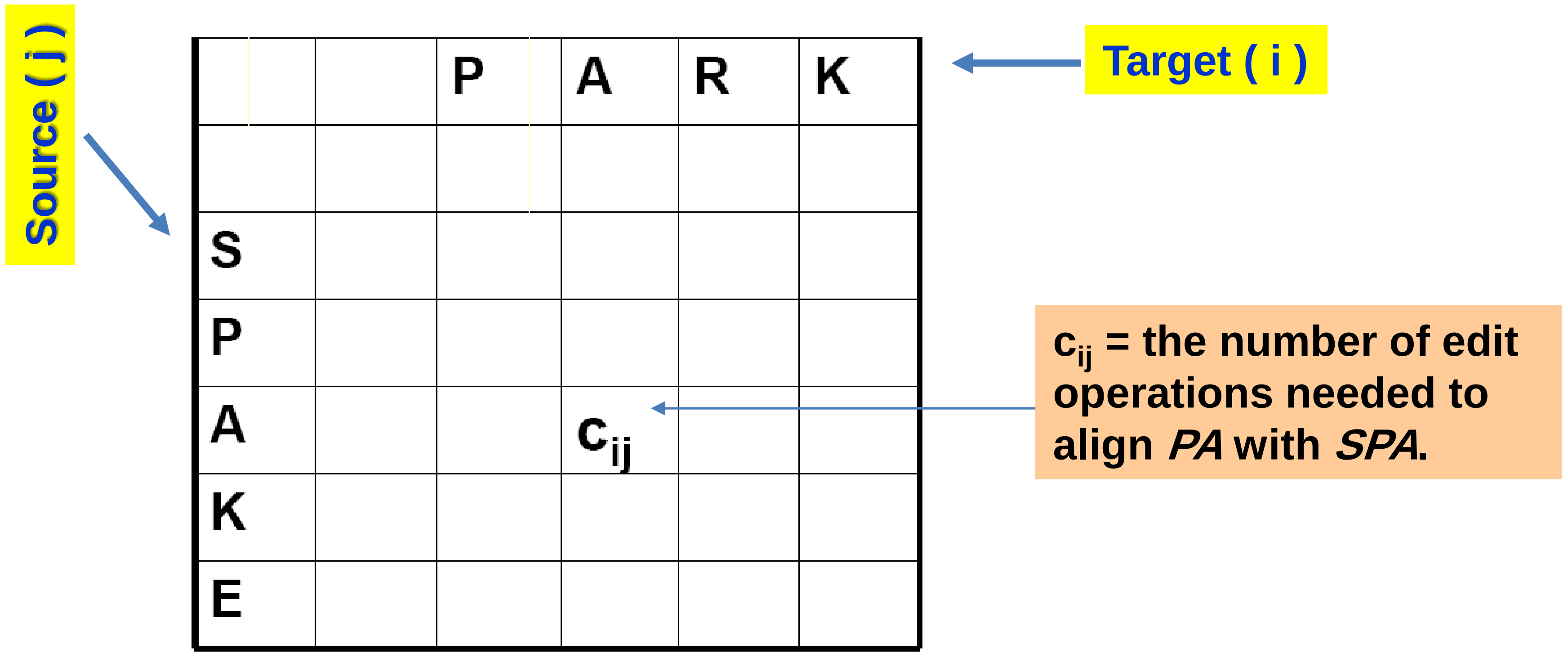
Target (i)

Deletion (LEFT)

- Using Levenshtein distance with cost of **1** for insertions or deletions, **2** for substitutions. Substitution of a character for itself has a cost of **0**.
- Using this version, the Levenshtein distance between intention and execution is 8.

Example: Minimum Edit Distance

- Edit operations for turning **SPAKE** into **PARK**
  - **Draw Matrix: Source Vs. Target**

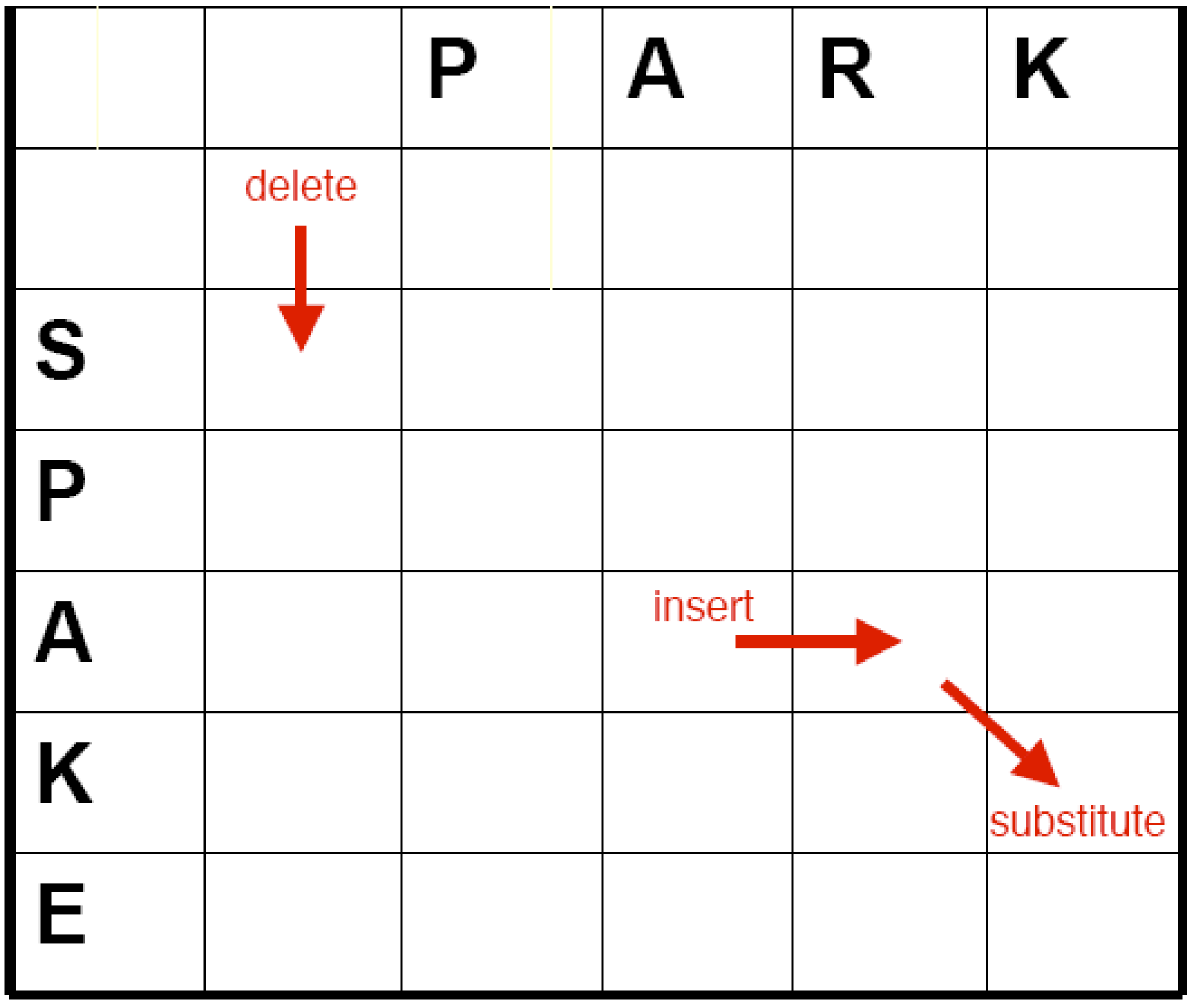


		P	A	R	K
S					
P					
A			c_{ij}		
K					
E					

c_{ij} = the number of edit operations needed to align *PA* with *SPA*.

Example: Minimum Edit Distance

- Measure distance between strings



		P	A	R	K
S					
P					
A					
K					
E					

Example: Minimum Edit Distance

		P	A	R	K
	c_{00}	c_{02}	c_{03}	c_{04}	c_{05}
S	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}
P	c_{20}	subst c_{21}	delete c_{22}	c_{23}	c_{24}
A	c_{30}	insert c_{31}	???		
K					
E					

D(i,j) = score of **best** alignment from *s1..si* to *t1..tj*

$$
= \min \begin{cases} D(i-1,j-1), \text{ if } si=tj & \textit{//copy} \\ D(i-1,j-1)+1, \text{ if } si!=tj & \textit{//substitute} \\ D(i-1,j)+1 & \textit{//insert} \\ D(i,j-1)+1 & \textit{//delete} \end{cases}
$$

Example: Minimum Edit Distance

- Initialize **Matrix**:

		P	A	R	K
	0	1	2	3	4
S	1				
P	2				
A	3				
K	4				
E	5				

Fill 1,2,… length of "PARK"

Fill 1,2,… length of "SPAKE"

Example: Minimum Edit Distance Algorithm

- Filling in **Matrix …**

- Cost of Delete = 1

  Cost of Insert = 1

  Cost of Substitute =1

D(i,j) = score of **best** alignment from *s1..si* to *t1..tj*

$$= \min \begin{cases} D(i-1,j-1)+d(si,tj) & \textit{//substitute} \\ D(i-1,j)+1 & \textit{//insert} \\ D(i,j-1)+1 & \textit{//delete} \end{cases}$$

		P	**A**	**R**	**K**
	0	1	2	3	4
S	1	1			
P	2				
A	3				
K	4				
E	5				

Example: Minimum Edit Distance

- Filling in **Matrix** ...
- Cost of Delete = 1

  Cost of Insert = 1

  Cost of Substitute =1

D(i,j) = score of **best** alignment from *s1..si* to *t1..tj*

$$= \min \begin{cases} D(i-1,j-1)+d(si,tj) & \textit{//substitute} \\ D(i-1,j)+1 & \textit{//insert} \\ D(i,j-1)+1 & \textit{//delete} \end{cases}$$

		P	A	R	K
	0	1	2	3	4
S	1	1	2	3	4
P	2				
A	3				
K	4				
E	5				

Example: Minimum Edit Distance

- Filling in **Matrix** …

- Cost of Delete = 1

  Cost of Insert = 1

  Cost of Substitute =1

D(i,j) = score of **best** alignment from *s1..si* to *t1..tj*

$$= \min \begin{cases} D(i-1,j-1)+d(si,tj) & \textit{//substitute} \\ D(i-1,j)+1 & \textit{//insert} \\ D(i,j-1)+1 & \textit{//delete} \end{cases}$$

		P	**A**	**R**	**K**
	0	1	2	3	4
S	1	1	2	3	4
P	2	1			
A	3				
K	4				
E	5				

Example: Minimum Edit Distance

- Filling in **Matrix** …
- Cost of Delete = 1

  Cost of Insert = 1

  Cost of Substitute =1

D(i,j) = score of **best** alignment from *s1..si* to *t1..tj*

$$
= \min \begin{cases} D(i-1,j-1)+d(si,tj) & \textit{//substitute} \\ D(i-1,j)+1 & \textit{//insert} \\ D(i,j-1)+1 & \textit{//delete} \end{cases}
$$

		P	**A**	**R**	**K**
	0	1	2	3	4
S	1	1	2	3	4
P	2	1	2	3	4
A	3	2	1	2	3
K	4	3	2	2	2
E	5	4	3	3	3

Final cost of aligning all of both strings.

Minimum Edit Distance Algorithm

- **Create Matrix**
- Initialize 1 – length in LH column and bottom row
- For each cell

Take the minimum of:

  - Deletion: +1 from left cell
  - Insertion: +1 from cell below
  - Substitution: Diagonal +0 if same +2 if different

- Keep track of where you came from

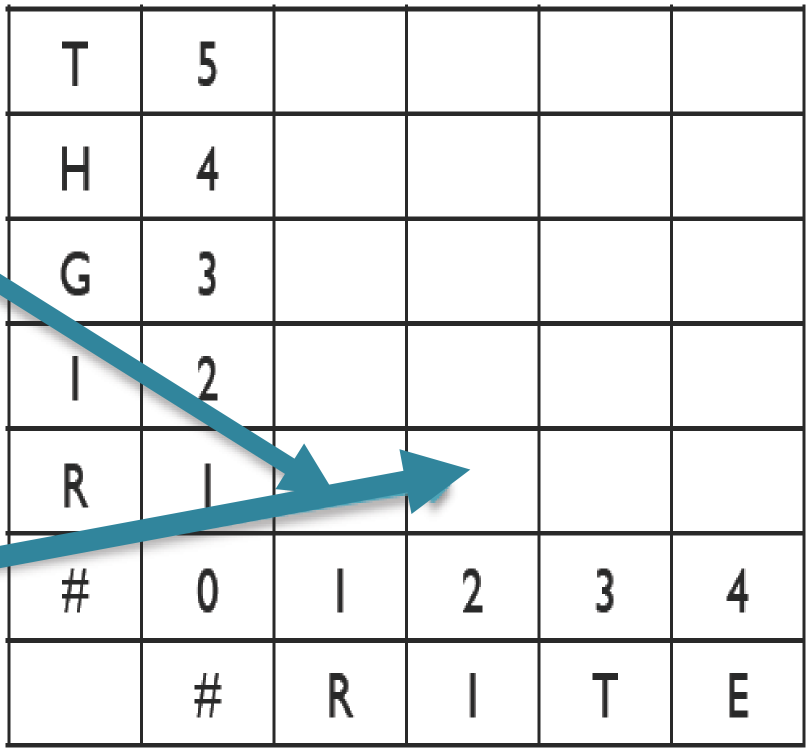
Example

- Computing minimum edit distances table between **Right** and **Rite**

Example …

- Minimum of:
  - 1+1 (left right)
  - 1+1 (bottom up)
  - 0+0 (diagonal)

- Minimum of:
  - 0+1 (left right)
  - 2+1 (bottom up)
  - 1+2 (diagonal)



T	5				
H	4				
G	3				
I	2				
R	1				
#	0	1	2	3	4
	#	R	I	T	E

Example …

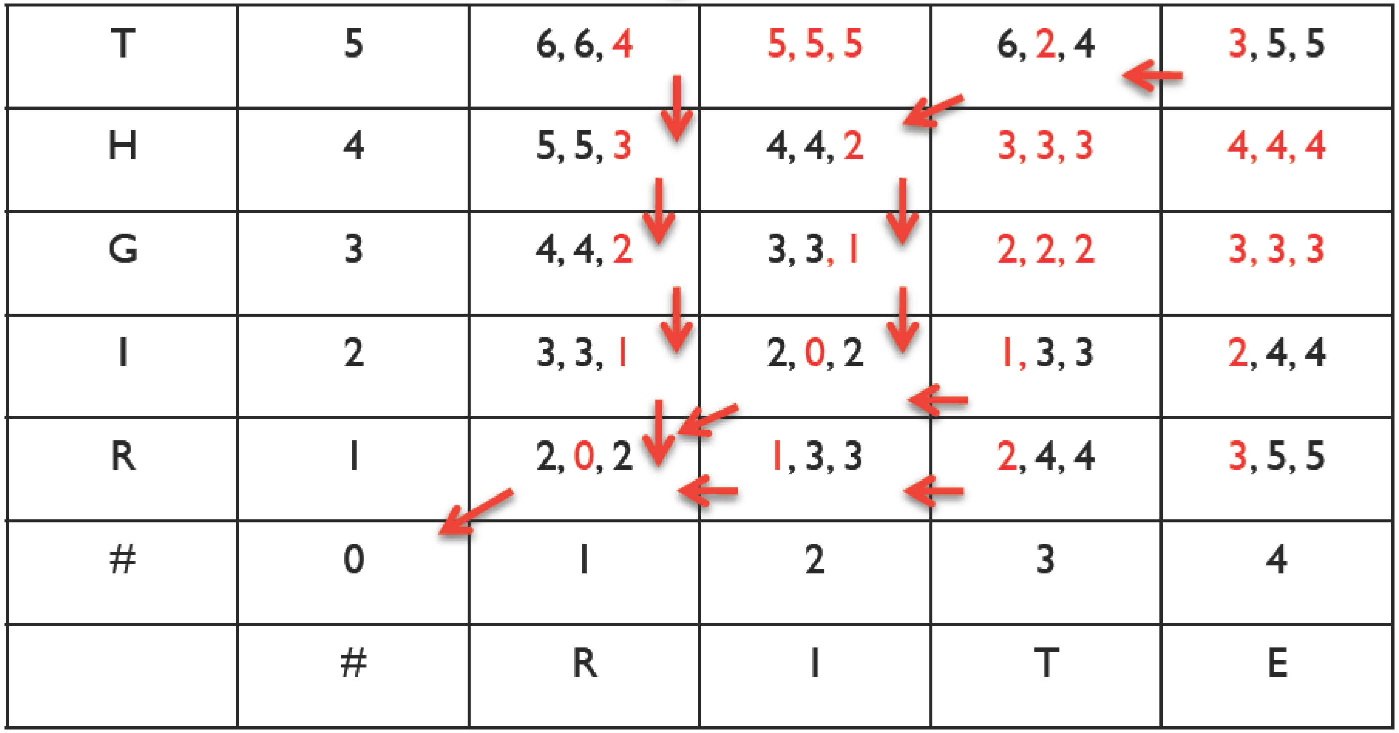
<u>In each box X, Y, Z values are</u>

X: From left: Insert-add one from left box

Y: Diagonal, Compare-0 if same, 2 if different

Z: From below: Delete-add one from lower box

Minimum is highlighted in red with arrow to source



T	5	6, 6, **4**	**5, 5, 5**	6, **2**, 4	**3**, 5, 5
H	4	5, 5, **3**	4, 4, **2**	**3, 3, 3**	**4, 4, 4**
G	3	4, 4, **2**	3, 3, **1**	**2, 2, 2**	**3, 3, 3**
I	2	3, 3, **1**	2, **0**, 2	**1**, 3, 3	**2**, 4, 4
R	1	2, **0**, 2	**1**, 3, 3	**2**, 4, 4	**3**, 5, 5
#	0	1	2	3	4
	#	R	I	T	E

Question?

- Computing minimum edit distances table by hand, figure out whether ***drive*** is closer to ***brief*** or to ***divers***, and what the edit distance is. You may use any version of distance that you like.

- Write a program to do the previous task.

Language Modeling

- Back to word prediction
- We can model the word prediction task as the ability to assess the **conditional probability** of a word given the previous words in the sequence

$$P(w_n|w_1,w_2\ldots w_{n-1})$$

- We'll call a statistical model that can assess this a Language Model

Language Modeling

- How might we go about calculating such a conditional probability?
  - One way is to use the definition of conditional probabilities and look for counts. So to get
  - **P(the | its water is so transparent that)**
- By definition that's

$$\frac{P(\text{its water is so transparent that the})}{P(\text{its water is so transparent that})}$$

- We can get each of those from counts in a large corpus.

Easy Estimate

- How to estimate?

  P(the | its water is so transparent that)

$$P(\text{the} \mid \text{its water is so transparent that}) = \frac{\text{Count(its water is so transparent that the)}}{\text{Count(its water is so transparent that)}}$$

Language Modeling

- Unfortunately, for most sequences and for most text collections we won’t get good estimates from this method.
- What we’re likely to get is **0. Or worse 0/0**.
- Clearly, we’ll have to be a little more clever.
- Let’s use
  - the chain rule of probability
  - a particularly useful independence assumption.

The Chain Rule

- Recall the definition of conditional probabilities
- Rewriting:

$$P(A|B) = \frac{P(A \wedge B)}{P(B)}$$

$$P(A \wedge B) = P(A|B)P(B)$$

- For sequences...

P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)

- In general

P(x1,x2,x3,…xn) = P(x1)P(x2|x1)P(x3|x1,x2)…P(xn|x1… xn-1)

The Chain Rule

$$\begin{aligned} P(w_1^n) &= P(w_1)P(w_2|w_1)P(w_3|w_1^2)\ldots P(w_n|w_1^{n-1}) \\ &= \prod_{k=1}^{n} P(w_k|w_1^{k-1}) \end{aligned}$$

P(its water was so transparent)=

P(its)*

P(water | its)*

P(was | its water)*

P(so | its water was)*

P(transparent | its water was so)

Unfortunately

- There are still a lot of possible sentences
- In general, we'll never be able to get enough data to compute the statistics for those longer prefixes
  - Same problem we had for the strings themselves

Independence Assumption

- Make the simplifying assumption

  P(lizard| the,other,day,I,was,walking,along,and,saw,a)

  = P(lizard|a)
- Or maybe

  P(lizard|

  the,other,day,I,was,walking,along,and,saw,a)

  = P(lizard|saw,a)
- That is, the probability in question is independent of its earlier history.

Independence Assumption

- This particular kind of independence assumption is called a **Markov assumption**
- So for each component in the product replace with the approximation (assuming a prefix of N)

$$P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-N+1}^{n-1})$$

- Bigram version

$$P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-1})$$

Estimating Bigram Probabilities

- The Maximum Likelihood Estimate (MLE)

$$P(w_i \mid w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

Example

- <s> I am Sam </s>
- <s> Sam I am </s>
- <s> I do not like green eggs and ham </s>

$P(\texttt{I} \mid \texttt{<s>}) = \frac{2}{3} = .67 \qquad P(\texttt{Sam} \mid \texttt{<s>}) = \frac{1}{3} = .33 \qquad P(\texttt{am} \mid \texttt{I}) = \frac{2}{3} = .67$

$P(\texttt{</s>} \mid \texttt{Sam}) = \frac{1}{2} = 0.5 \qquad P(\texttt{Sam} \mid \texttt{am}) = \frac{1}{2} = .5 \qquad P(\texttt{do} \mid \texttt{I}) = \frac{1}{3} = .33$

Maximum Likelihood Estimates

- The maximum likelihood estimate of some parameter of a model M from a training set T
- Is the estimate that maximizes the likelihood of the training set T given the model M

Maximum Likelihood Estimates

- Suppose the word Chinese occurs 400 times in a corpus of a million words (Brown corpus)
- What is the probability that a random word from some other text from the same distribution will be "Chinese"
  - **MLE estimate is 400/1000000 = .004**
- This may be a bad estimate for some other corpus
- But it is the **estimate that makes it most likely that** "Chinese" will occur 400 times in a million word corpus.

Example
(Berkeley Restaurant Project Sentences)

– P(I want to eat Chinese food) = P(I | <start>) P(want | I) P(to | want) P(eat | to) P(Chinese | eat) P(food | Chinese)

A Bigram Grammar Fragment from BERP

eat on	.16	eat Thai	.03
eat some	.06	eat breakfast	.03
eat lunch	.06	eat in	.02
eat dinner	.05	eat Chinese	.02
eat at	.04	eat Mexican	.02
eat a	.04	eat tomorrow	.01
eat Indian	.04	eat dessert	.007
eat today	.03	eat British	.001

Estimating Bigram Probabilities

$$P(w_i \mid w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

Additional Grammar

<start> I	.25	Want some	.04
<start> I’d	.06	Want Thai	.01
<start> Tell	.04	To eat	.26
<start> I’m	.02	To have	.14
I want	.32	To spend	.09
I would	.29	To be	.02
I don’t	.08	British food	.60
I have	.04	British restaurant	.15
Want to	.65	British cuisine	.01
Want a	.05	British lunch	.01

Computing Sentence Probability

- P(I want to eat British food) = P(I|<start>) P(want|I) P(to|want) P(eat|to) P(British|eat) P(food|British)

  = .25×.32×.65×.26×.001×.60 = .000080
- Vs.
- P(I want to eat Chinese food) = .00015
- Probabilities seem to capture “syntactic'' facts, “world knowledge''
  - eat is often followed by a NP
  - British food is not too popular
- N-gram models can be trained by counting and normalization

Kinds of Knowledge

- N-gram probabilities capture a range of interesting facts about language.

P(english | want) = .0011	**World knowledge**
P(chinese | want) = .0065	
P(to| want) = .66	**Syntax**
P(eat | to) = .28	
P(food | to) = 0	
P(want | spend) =0	
P (I | <s>) = .25	**Discourse**

N-grams Issues

- Sparse data
  - Not all N-grams found in training data, need smoothing
- Change of domain

- N-grams more reliable than (N-1)-grams
  - But even more sparse
    - Generating Shakespeare sentences with random unigrams...
      - Every enter now severally so, let
    - With bigrams...
      - What means, sir. I confess she? then all sorts, he is trim, captain.
    - Trigrams
      - Sweet prince, Falstaff shall die.

Problem

- Let's assume we're using N-grams
- How can we assign a probability to a sequence where one of the component n-grams has a value of zero.

- Assume all the words are known and have been seen:
  - Go to a lower order n-gram
  - Back off from bigrams to unigrams
  - Replace the zero with something else