Processamento Estatístico da Linguagem Natural

Aula 6
Professora Bianca
(Sala 302 – Bloco E)
bianca@ic.uff.br
http://www.ic.uff.br/~bianca/peln/

Correção Ortográfica

- Três tipos de problemas:
  - Detecção de palavras que não existem.
    - Detectar que a palavra *Graffe* não existe em inglês.
  - Correção de palavras isoladas.
    - Corrigir *graffe* para *giraffe*
  - Detecção e correção de erros dependentes de contexto.
    - Corrigir *three* pra *there* na frase "*I went three*".

Detecção de palavras que não existem

- Podemos usar um parser morfológico (FST) para reconhecer as palavras da língua.
  - Se o parser não reconhece ela não existe.
- Lida automaticamente com as inflexões e derivações morfológicas.
  - Não é necessário listar todas as palavras da língua explicitamente.

Correção de Erros

- Temos que encontrar a palavra que tem mais chance de ser a palavra original que foi escrita incorretamente.
- Precisamos de uma métrica para calcular a distância entre a palavra "superficial" (digitada) e cada possível palavra "fonte" (do dicionário).
  - Soluções atuais usam probabilidade e serão tratadas no capítulo 4.
    - Mas são baseadas no conceito de **distância mínima de edição** que será discutido nesse capítulo.

Distância Mínima de Edição

- É o número mínimo de edições (inserções, deleções e substituições) necessário para transformar uma string em outra.
  - Também chamada de medida de Levenshtein

```
INTE*NTION
||||||||||
*EXECUTION
d s s  i s
```

Alinhamento

Distância Mínima de Edição

- A distância mínima pode ser computada usando um algoritmo de **Programação Dinâmica (PD)**.
  - Classe de algoritmos introduzida por Bellman que resolve problemas combinando soluções de sub-problemas.
    - Usa uma tabela para armazenar resultados de sub-problemas.
    - Dessa forma ao invés de repetir a computação, pode-se olhar o valor na tabela.

PD para Distância Mínima de Edição

- Solução parcial = Custo de alinhar ***s*** até a posição ***i*** com ***t*** até a posição ***j***.
- Próximo passo = Para alinhar ***s*** até a posição ***x*** e ***t*** até a posição ***y***, a próxima operação deve ser substituição, inserção ou deleção?

Tabela de PD para Distância de Edição

Medir distância entre as strings PARK

SPAKE

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

c_{ij} = número de operações de edição necessárias para alinhar PA com SPA

Tabela de PD para Distância de Edição

Medir distância entre as strings PARK

SPAKE

Operações de edição para transformar SPAKE em PARK

		P	A	R	K
	delete				
S					
P					
A			insert		
K					substitute
E					

Tabela de PD para Distância de Edição

Medir distância entre as strings PARK

SPAKE

		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}	c_{21}	c_{22}	c_{23}	c_{24}
A	c_{30}	c_{31}	???		
K					
E					

Tabela de PD para Distância de Edição

		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}	c_{21} (copy)	c_{22} (delete)	c_{23}	c_{24}
A	c_{30}	c_{31} (insert)	???		
K					
E					

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

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

Cálculo da Distância de Edição

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

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

(simplify by letting d(c,d)=0 if c=d, 1 else)

also let D(i,0)=i *(for i inserts)* and D(0,j)=j

Inicialização da tabela

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

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

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

Preenchendo a tabela

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

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

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

Preenchendo a tabela

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

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

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

Preenchendo a tabela

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

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

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

Preenchendo a tabela

		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

Custo total de alinhar as duas strings até o final.

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

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

Recuperando o alinhamento (Backtrace)

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

A *trace* indicates where the min value came from, and can be used to find edit operations and/or a best *alignment* (may be more than 1)

		C	O	H	E	N
	0	1	2	3	4	5
M	1	1	2	3	4	5
C	2	2	3	3	4	5
C	3	3	2	3	4	5
O	4	4	3	2	3	4
H	5	5	4	3	3	3
N						

Distância Mínima de Edição

- Algoritmo e código em diversas linguagens disponível em http://www.merriampark.com/ld.htm

Segunda Lista de Exercícios

- Capítulo 3 (segunda edição)
  - 3.4
  - 3.7
  - 3.10
  - 3.12
- Data de entrega: 02/10

Linguagem Python

- Introdução
  - Características de Python
  - Rodando programas
  - Módulos
- Tipos básicos
  - Números e variáveis
  - Strings
  - Listas e tuplas
  - Dicionários
- Fluxo de controle
  - Execução condicional
  - Repetições

Características de Python

- Gratuita. Roda em muitas plataformas.
- Fácil de ler.
  - Ao contrário de Perl = "write only language"
- Tempo de implementação rápido.
  - Ao contrário de Java.
- Orientada a objeto.
- NLTK = Natural Language Toolkit

Usando Python no modo interativo

- Usuário digitando em vermelho, máquina respondendo em preto.

```
$ python
>>> print "Hello everyone!"
Hello everyone!
>>> print 2+2
4
>>> myname = "Andrew"
>>> myname
'Andrew'
```


Módulos

- Arquivos texto contendo código Python.

foo.py

```
print 25*3                          # multiply by 3
print 'CompLing ' + 'lecture 3'     # concatenate with +
myname = 'Andrew'
```

Módulo sendo executado a partir da linha de comando:

```
$ python foo.py
75
CompLing lecture 3
$
```

Importando módulos

- Todo arquivo terminando em **`.py`** é um módulo python que pode ser importado.
  - A terminação **`.py`** é omitida.
  - As declarações do módulo (funções, variáveis) ficam disponíveis como atributos de um objeto que tem o nome do módulo.

```
$ python
>>> import foo
75
CompLing lecture 3
>>> foo.myname
'Andrew'
```


Recarregando módulos

- A importação de módulos é cara computacionalmente e Python só a realiza uma vez (mesmo que o arquivo seja editado).
- Para forçar que o arquivo seja importado novamente, devemos usar o comando "reload".

Edit foo.py to print 25*4 (instead of 25*3) and reload

```
>>> reload(foo)
100
CompLing lecture 3
<module 'foo' from 'foo.py'>
```


Atributos de Módulos

- Considere o arquivo **`bar.py`**

```
university = 'UMass'
department = 'Linguistics'
```

```
>>> import bar
>>> print bar.department
Linguistics

>>> from bar import department
>>> print department
Linguistics

>>> from bar import *
>>> print university
UMass
```


Estruturas de Programas em Python

- Programas são compostos de módulos.
- Módulos contém comandos.
- Comandos contém expressões.
- Expressões criam e processam objetos.

- Comandos incluem:
  - Atribuição de variáveis
  - Chamadas de função
  - Controle de fluxo
  - Declaração de função
  - Declaração de objeto
  - Leitura e impressão

Objetos "built-in" de Python

- Números: inteiro e ponto-flutuante
- Strings
- Listas
- Dicionários
- Tuplas
- Arquivos

Expressões numéricas e variáveis

- Operadores usuais: +, *, /, **
- Precedência usual:
  - A * B + C * D = (A*B) + (C*D)
- Módulos úteis: **`math`** e **`random`**

- Variáveis
  - Criadas na primeira atribuição
  - São substituídas pelo seu valor quando usadas em expressões
  - Tem que ter recebido um valor antes do primeiro uso.
  - Não precisam ser declaradas