1 Introducción

Un modelo del lenguaje estadístico asigna una probabilidad a una secuencia de m palabras $P(w_1, \ldots, w_m)$ mediante una distribución de probabilidad. Tener una forma de estimar la verosimilitud de diferentes frases es útil en muchas aplicaciones de procesamiento de lenguaje natural. Modelación del lenguaje se utiliza en el reconocimiento de voz, traducción automática, etiquetado de discurso, análisis, reconocimiento de escritura, la recuperación de información y otras aplicaciones. [Wikipedia]

Una de las aplicaciones particulares de los modelos de lenguaje es la corrección ortográfica y gramatical. Por ejemplo, se puede utilizar para detectar los famosos "errores de dedo" y para mejorar la redacción en los documentos. Por ejemplo, en el siguiente tutorial de Norvig se aborda la corrección de "errores de dedo": http://norvig.com/spell-correct.html utilizando una estrategia buscar los candidatos c más probables dada la palabra escrita w, es decir $argmax_{c \in candidates} P(c|w)$. Estudie y comprenda el funcionamiento de la estrategia propuesta por Norvig para solucionar los siguientes puntos de este problema. Siéntete libre de adaptar y/o extender el código de Norvig para completar cualquier punto.

2 La siguiente palabra más probable

Construya una función que utilizando el archivo big.txt del Tutorial de Norvig, estime la siguiente palabra más probable dada una anterior. Formalmente, dada una palabra w_i, la función debe calcular w_{i+1} tal que:

$$w_{i+1} = argmax_{w_{i+1}} P(w_{i+1}|w_i) \tag{1}$$

Notas: 1) Podemos asumir que ambas palabras siempre existirán en la colección. 2) Se requiere una función P similar a la de Norvig, pero que también estime la probabilidad condicional de cada palabra. Por ejemplo la nueva función $PCond("york","new")$ debería estimar $P("york"|"new")$

```
def next_word(word):
    # your code here
    return ''
```

2.1 El ahorcado

Diseñe una función que sea capaz de encontrar los caracteres faltantes de una palabra. Sea creativo para extender y mejorar de algún modo la función propuesta por Norvig. La función debe trabajar como sigue:

```
>>> hangman("pe_p_e")
'people'

>>> hangman("phi__sop_y")
'philosophy'

>>> hangman("si_nif_c_nc_")
'significance'

>>> hangman("kn__l_d_e")
'knowledge'
```

2.2 El ahorcado al extremo

Combine el uso de su función en el problema de la Sección 2.1 (la siguiente palabra más probable) con la función de Norvig para completar palabras del ahorcado con mayor precisión utilizando una palabra del contexto. La función debería trabajar como sigue:

```
>>> hangman("united", "_t_t__")
'states'

>>> hangman("white","___s_")
'house'

>>> hangman("new","___k")
'york'

>>> hangman("abraham", "l_____n")
'lincoln'
```

Tips y Consideraciones:

- Use su creatividad para adaptar o exter las estrategias del tutorial de Norvig. Existen muchas soluciones simples, pero SOLO UNAS POCAS CREATIVAS y EFECTIVAS.
- La función de el ahoracado debe poder tratar con hasta 4 caracteres desconocidos en palabras de longitud arbitraria, y la del ahorcado extremo hasta con una conocida. Además la función debe trabajar en tiempo razonable (máximo 1 minuto en una laptop).
- Puede utilizar otros algoritmos para obtener distancias de edición (e.g., Levenshtein) o bien para subcadenas (e.g., Karp Rabin, Aho-Corasick). ¡Deje volar su imaginación!

Explica tu estrategia, muestre que su propuesta funciona con diferentes ejemplos, y discuta el resultado.

3 Corrección ortográfica simple

Utilice las siguientes funciones de ayuda para leer el texto en un string y lista de palabras.

```
import os
import re

# simple extraction of words
def words(text): return re.findall(r'\w+', text.lower())

# siple loading of the documents
from keras.preprocessing.text import Tokenizer
def get_texts_from_catdir(cat_dir):
    texts = []
    TARGET_DIR = cat_dir # "./target"
    for f_name in sorted(os.listdir(TARGET_DIR)):
        f_path = os.path.join(TARGET_DIR, f_name)
        #print(f_name)
        #print(f_path)
        f = open(f_path, "r",  encoding="utf8")
        print(f_name)
        texts += [f.read()]
        f.close()
    print("%d files loaded." % len(texts))
    return texts

# Load the RAW text
target_txt = get_texts_from_catdir("./target")

# Print first 10 words in document0
print(words(target_txt[0])[:10])
```

Instrucciones: En esta parte necesitará construir un corrector ortográfico simple y generador de candidatos. Para ello descargue los archivos de apoyo para esta pregunta de aquí: https://github.com/fagonzalezo/dl-tau-2017-2/blob/master/assign2_q2_files.zip. Descomprima el archivo y ponga su contenido en la raíz de su notebook o script de python. En este archivo se le proporcionan 10 noticias en el directorio "target" en las cuales se han introducido algunos errores. Además, para evaluar a su corrector también tiene el directorio golden que contiene el texto correcto. Siéntete libre de adaptar y/o extender el código de Norvig para tener un corrector básico y completar esta parte. **Es requisito indispensable usar alguna estrategia que combine la propuesta de Norvig con su función en la Sección 2.1 para tener una mejor estrategia que utilize el contexto.**

3.1 Mezclar WORDS[word] con DIC_WORDS_IN_NEWS

Genere un solo diccionario/vocabulario global mezclando los diccionarios DIC_WORDS_IN_NEWS y WORDS y sus frecuencias. Asegúrese que si una palabra de un diccionario ya está en el otro, entonces el diccionario resultante contendrá las suma de las frecuencias.

```
import json
with open('WORDS_IN_NEWS.txt', 'r') as infile:
    WORDS_IN_NEWS = json.load(infile)

def merge_dic(words_dic, words_in_news_dic):
    '''
    words_dic: for example WORDS
    words_in_news_dic: for example WORDS_IN_NEWS
    Returns: Merged dictionary merged_word_dic
    '''
    # Your code goes here
    return merged_word_dic

WORDS = merge_dic(WORDS, WORDS_IN_NEWS)
```

3.2 Detecte las palabras mal escritas en las noticias

Cree una funcion que dadas las palabras de un texto, regrese una lista conteniendo las tuplas de cada palabra que no está bien escrita y su lista de palabras candidatas para corregir. La función debe trabajar con palabras fuera del vocabulario en el diccionario y generar posibles candidatos de reemplazo. Sientase libre de adaptar código de Norvig.

```
def misspelled_and_candidates(target_words):
    '''
    target_words: a text with possible misspellings
    represented as a list of words
    Returns: list of tuples of the form
    [("misspelled-word-1", ["candidate-word-1_1",
    "candidate-word-1_2"]), ...]
    '''
    # Your code goes here
    return misspelled_candidates

# Example call to print the output. Target_txt[0] is the raw data from file 0.
print(misspelled_and_candidates(words(target_txt[0])))

# Print misspelled words and candidates for each document in
target_txt list
for text in target_txt:
    print(misspelled_and_candidates(words(text)))
```

3.3 Corrección completa

Combine el uso de su función next_word con su generador de candidatos, para diseñar una estrategia que realice una corrección ortográfica completa.

```
def spell_correction(input_text):
    '''
    input_text: a text with possible misspellings represented as a list of words
    Returns: the text with corrected misspellings
    '''
    # Your code goes here
    return corrected_words

# Apply the correction to each target text
corrected_txt = []
for t_txt in target_txt:
    corrected_txt += [spell_correction(words(t_txt))]
```

Explica tu estrategia, muestre que su propuesta funciona con diferentes ejemplos, y discuta el resultado. Aplique su corrector a las noticias y muestre que funciona para corregirlas. Finalmente desarrolle pruebas simples que muestren su estrategia completa trabajando. Abajo se proporcionan algunos ejemplos de pruebas y su salida esperada:

```
s=['i', 'hav', 'a', 'ham']
misspelled_and_candidates(s)
print(spell_correction(s))
Output: ['i', 'have', 'a', 'ham']
s=['my', 'countr', 'is', 'biig']
misspelled_and_candidates(s)
print(spell_correction(s))
Output: ['my', 'country', 'is', 'big']
s=['i','want','t00','eat']
misspelled_and_candidates(s)
print(spell_correction(s))
Output: ['i','want','to','eat']
s=['the','science','0ff','computer']
misspelled_and_candidates(s)
print(spell_correction(s))
Output: ['the','science','of','computer']
s=['the','science','off','computer']
misspelled_and_candidates(s)
print(spell_correction(s))
Output: ['the','science','of','computer']
s=['i','want','too','eat']
misspelled_and_candidates(s)
print(spell_correction(s))
Output: ['i','want','to','eat']
```