T4: Modelos de Lenguaje Estadísticos

CIMAT, Procesamiento de Lenguaje Natural, Ciencias de la Computación

Profesor: Dr. Adrián Pastor López Monroy

Entregar: Lunes 15 de Marzo de 2021 antes de las 23:59:59

1 Instrucciones

Realiza los siguientes puntos en un notebook de Python *lo mejor organizado y claro posible*. Ponga su nombre completo al archivo de entrega (e.g., adrian_pastor_lopez_monroy.ipynb) y también en la primera celda del notebook junto con el número de Tarea. Al entregar la tarea, sube al classroom el notebook como un archivo. El notebook deberá haber sido ejecutado en tú máquina y mostrar el resultado en las celdas. Para esta Tarea estudie y comprenda el funcionamiento de la estrategia propuesta por Norvig http://norvig.com/spell-correct.html. Siéntete libre de adaptar y/o extender parcial o totalmente el código de Norvig para esta tarea.

1.1 Se vale pedir ayuda y "copiar"

Se vale pedir ayuda y/o copiar con atribución entre los miembros de la clase y apegándose estrictamente a los siguientes puntos:

1. Del total de actividades que se solicitan hacer (8 con valor para esta tarea) solo puedes pedir ayuda/copiar en un total de **una**.
2. Para los puntos dónde se pide ayuda, brevemente escribe en qué pediste ayuda y a quién.
3. Si tuviste que reusar alguna parte de código que no es tuyo, deja claro dos cosas: 1) brevemente porque tuviste dificultad para hacerlo, 2) cómo lo solucionó tu compañero.
4. Se darán hasta 5pts/100pts extra en esta tarea (no son acumulables) si ayudaste a algún compañero y se comenta en ambas tareas. Se pueden ayudar mutuamente en diferentes puntos y ganar cada quién los 5pts/100pts extra. Siempre respetando el punto 1.

2 El ahorcado (10pts)

Diseñe una función que sea capaz de encontrar los caracteres faltantes de una palabra. Para ello proponga una adaptación simple de la estrategia de corrección ortográfica propuesta por Norvig. La función de el ahorcado debe poder tratar con hasta 4 caracteres desconocidos en palabras de longitud arbitraria. La función debe trabajar en tiempo razonable (≈ 1 minuto en una laptop). La función debe trabajar como sigue con 10 ejemplos:

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

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

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

Nota:

- Debe resolver este punto con una extensión simple de la estrategia de Norvig, aunque existan formas más eficientes con distancias de edición (e.g., Levenshtein) o de subcadenas (e.g., Karp Rabin, Aho-Corasick, Tries, etc.).

3 Modelo de Lenguaje (50pts)

3.1 Construcción de Modelo (15pts/50pts)

1. Construya un corpus para entrenar un modelo de lenguaje. Para ello use la parte de libros gutenberg del archivo big.txt, y combinelo con los datos de wikitex-2 o wikitex-103, según le convenga en tiempo y hardware. Defina su vocabulario y enmascare con *<unk>* toda palabra que no esté en su vocabulario (los wikitex ya lo tienen, pero tal vez necesite acotarlo más). Valore si conviene usar el tokenizador de oraciones de nltk, o usar el texto como una oración gigante (mencioné cuál utilizó y por qué). Para este ejercicio podría no ser necesario definir tokens especiales <s> y *</s>* dependiendo de si tokenizo oraciones. Limpie el texto dejando solo palabras en minúscula, quitando puntuación, números, etc.
2. Entrene tres modelos de lenguaje: $P_{unigramas}(w_1^n)$, $P_{bigramas}(w_1^n)$, $P_{trigramas}(w_1^n)$. Para cada uno proporcione una interfaz (función) sencilla para $P_{n-grama}(w_1^n)$ y $P_{n-grama}(w_1^n|w_{n-N+1}^{n-1})$. Los modelos deben tener una estrategia común para lidiar con secuencias no vistas. Puede optar por un suavizamiento *Laplace* o un *Good-Turing discounting*.
3. Construya un modelo interpolado con valores λ fijos:

$$\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n|w_{n-2}w_{n-1}) + \lambda_2 P(w_n|w_{n-2}w_{n-1}) + \lambda_3 P(w_n)$$

   Para ello experimente con el modelo en particiones de 80%, 10% y 10% para entrenar (*train*), ajuste de parámetros (*val*) y prueba (*test*) respectivamente. Muestre como bajan o suben las perplejidades en validación, finalmente pruebe una vez en test. Explore algunos valores $\vec{\lambda}$ y elija el mejor, i.e., [1/3, 1/3, 1/3], [.4, .4, .2], [.2, .4, .4], [5, 4, 1] y [.1, .4, .5].

3.2 Corrección ortográfica y gramatical (15pts/50pts)

Implemente una función *spell_correction* que combine el uso del generador de candidatos de Norvig con el modelo de lenguaje entrenado en el punto anterior. La idea es detectar el error ortográfico, generar candidatos, **pero diseñar alguna estrategia para dejar que el modelo de lenguaje elija el mejor candidato para aplicar la corrección.** Sí alguna palabra de entrada no está en su vocabulario reemplácela por **<unk>**.

Explica en que consiste tu estrategia; **máximo dos párrafos**. Escoge algunos ejemplos que ilustren el comportamiento de su estrategia. Abajo se proporcionan algunas sugerencias de pruebas simples y su salida esperada, aunque puede usar otras si así lo desea.

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

# Apply the correction to some examples that you chose
# corrected_txt = [text1, text2, ... etc]
for t_txt in target_txt:
    corrected_txt += [spell_correction(words(t_txt))]
    # Write some code for to print the following:
    # 1) The original word list
    # 2) The corrected word list
    # 3) The best candidate according to the spell corrector
    # 4) The best candidate according to your language model
```

Ejemplos ilustrativos[1]:

```
s=['i', 'hav', 'a', 'ham']
print(misspelled_and_candidates(s))
print(grammar_candidates(s))
print(spell_correction(s))
Output: ['i', 'have', 'a', 'ham']

s=['my', 'countr', 'is', 'biig']
print(misspelled_and_candidates(s))
print(grammar_candidates(s))
print(spell_correction(s))
Output: ['my', 'country', 'is', 'big']

s=['i','want','t00','eat']
print(misspelled_and_candidates(s))
print(grammar_candidates(s))
print(spell_correction(s))
Output: ['i','want','to','eat']

s=['the','science','0ff','computer']
print(misspelled_and_candidates(s))
print(grammar_candidates(s))
print(spell_correction(s))
Output: ['the','science','of','computer']
```

3.3 Corrección gramatical (15pts/50pts)

Para el escenario en el que no hay errores ortográficos, proponga una estrategia simple para **sugerir** corrección gramatical. Discuta en no más de **dos párrafos** las ventajas/desventajas

[1]Nota: La función *misspelled_and_candidates* debe imprimir los candidatos para el corrector ortográfio. Mientras la función *grammar_candidates* debe imprimir los candidatos a corrección según el modelo de lenguaje.

de su estrategia. Escoja 3 ejemplos para ilustrar dónde su estrategia funciona y 3 en los que no.

```
Ejemplo ilustrativo:
s=['the','science','off','maths']
print(badgrammar_and_candidates(s))
# Here you have to show the candidates for the languague model
```

3.4 Auto-completado (5pts/50pts)

Use su modelo de lenguaje para diseñar una función que sugiera la siguiente palabra dada una secuencia previa. Por ejemplo:

```
s=['i','would','like','to', 'visit', 'new']
print(autocomplete_and_candidates(s))
Output: s=['i','would','like','to', 'visit', 'new', 'york']
# Show other five top candidates
```

Escoja 3 ejemplos para ilustrar dónde su estrategia funciona y 3 en los que no. En **un párrafo** explique brevemente porque falló.

4 Generación de Texto (40pts)

Para esta parte reentrenará su modelo de lenguaje interpolado:

$$\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n|w_{n-2}w_{n-1}) + \lambda_2 P(w_n|w_{n-2}w_{n-1}) + \lambda_3 P(w_n)$$

Realice las siguientes actividades:

1. Para los tuits de agresividad use solo palabras en minúscula, quite signos de puntuación, números, etc. Agregue tokens especiales de <s> y *</s>* a cada "tuit".
2. **(20pts/40pts)** Proponga una estrategia con base en *Expectation Maximization* para encontrar los valores de interpolación en $\hat{P}_{agresivo}$ usando todo el dataset de agresividad. Para ello experimente con el modelo en particiones de 80%, 10% y 10% para entrenar (*train*), ajuste de parámetros (*val*) y prueba (*test*) respectivamente. [2] Muestre como bajan las perplejidades en 5 iteraciones que usted elija (de todas las que sean necesarias de acuerdo a su EM) en validación, y pruebe una vez en test. Si no logró diseñar algo basado en EM **(-10pts)** explore algunos valores $\vec{\lambda}$ en algunas pruebas y elija el mejor, por ejemplo: [1/3, 1/3, 1/3], [.4, .4, .2], [.2, .4, .4], [5, 4, 1] y [.1, .4, .5].
3. **(10pts/40pts)** Haga una función "tuitear" basado en su modelo de lenguaje $\hat{P}_{agresivo}$ del último punto. El modelo deberá poder parar *automáticamente* cuando genere algún n-grama con el símbolo de terminación de tuit al final (e.g., "</s>"), o 50 palabras. Proponga algo para que en los últimos tokens sea más probable generar el token "</s>").
4. **(10pts/40pts)** Use la intuición que ha ganado en esta tarea y los datos de las mañaneras para entrenar un modelo de lenguaje *AMLO*. Haga una un función "dar_conferencia()". Puede entrenar su modelo con un solo string gigante de todas las intervenciones de AMLO, sin tokens "<s>" y "</s>". Generé un discurso de 300 palabras y detenga al modelo de forma abrupta.

[2] Note que cada partición tiene tanto tuits agresivos y no agresivos