Prediccion_de_secuencias_de_palabras

September 16, 2023

Predicción de secuencias de palabras

Emmanuel Santos Rodríguez

1 Introducción

Los modelos de lenguaje nos permiten asignar probabilidades a las secuencias de palabras. Mediante estos modelos, podemos predecir cuál será la siguiente palabra en una secuencia dada o obtener la probabilidad de toda una oración. Los n-gramas, que son secuencias de n palabras, son el modelo más sencillo de este tipo. En particular, usando bigramas podemos aproximar la probabilidad de una palabra dadas todas las palabras previas utilizando solo la probabilidad condicional de la palabra anterior. Para estimar las probabilidades para los bigramas se usa el estimador de máxima verosimilitud (MLE). Obtenemos el MLE para los parámetros de un modelo de n-gramas obteniendo recuentos de un corpus y normalizando los recuentos para que se encuentren entre 0 y 1.

El propósito de este documento es realizar la predicción de secuencias de palabras utilizando un modelo de bigramas con elestimador de máxima verosimilitud con y sin suavizado para un corpus de textos en español.

2 Desarrollo

2.1 Descripción de los datos

Para el desarrollo de esta solución se utilizó un corpus de documentos de una conferencia del parlamento europeo. Cada línea es un párrafo que considera a nivel de oración. A continuación se muestra una oración ejemplo del corpus:

> yo como austríaco , pero creo que todos nosotros tenemos aún un vivo recuerdo de la catástrofe que el año pasado costó la vida a numerosas personas en el túnel del tauern y en el que después hubo que reconstruir durante largos meses y con un gigantesco gasto financiero lo que fue destruido por el incendio en el túnel .

2.2 Implementación

Para la implementación de esta solución se utilizó Python. A continuación se muestra el código para los puntos más importantes.

2.2.1 Preprocesamiento

Antes de construir nuestro modelo de n-gramas, en particular bigramas, debemos realizar un procesamiento para el corpus descrito anteriormente. Este procesamiento consiste en convertir a minúsculas, eliminar puntuación y añadir marcadores de inicio y final para cada oración. En este caso, usamos <s> como marcador de inicio y </s> como marcador de final. Al estar conformado por documentos oficiales, se considera que el corpus está bien escrito. Es importante recalcar que se considera cada línea como una oración sin necesidad de encontrar las oraciones por cada párrafo.

[ ]:
```
def preprocess(text):
    #convertir a minuscula y remover signos de puntuacion
    text_lower = text.lower().translate(str.maketrans('', '', string.
 ↪punctuation))
    delimited_text = f"<s> {text_lower} </s>"
    #obtenemos los tokens
    text_tokens = delimited_text.split()
    return text_tokens
```

La función anterior se encarga de convertir el texto a minúscula, remover los signos de puntuación y añadir los marcadores de inicio y fin de oración así como dividir el texto en tokens.

De esta forma, después de aplicar el procesamiento a la oración de ejemplo anterior, obtenemos lo siguiente:

> <s> yo como austríaco pero creo que todos nosotros tenemos aún un vivo recuerdo de la catástrofe que el año pasado costó la vida a numerosas personas en el túnel del tauern y en el que después hubo que reconstruir durante largos meses y con un gigantesco gasto financiero lo que fue destruido por el incendio en el túnel </s>

y al dividirlo en tokens se obtiene:

[‘<s>’, ‘yo’, ‘como’, ‘austríaco’, ‘pero’, ‘creo’, ‘que’, ‘todos’, ‘nosotros’, ‘tenemos’, ‘aún’, ‘un’, ‘vivo’, ‘recuerdo’, ‘de’, ‘la’, ‘catástrofe’, ‘que’, ‘el’, ‘año’, ‘pasado’, ‘costó’, ‘la’, ‘vida’, ‘a’, ‘numerosas’, ‘personas’, ‘en’, ‘el’, ‘túnel’, ‘del’, ‘tauern’, ‘y’, ‘en’, ‘el’, ‘que’, ‘después’, ‘hubo’, ‘que’, ‘reconstruir’, ‘durante’, ‘largos’, ‘meses’, ‘y’, ‘con’, ‘un’, ‘gigantesco’, ‘gasto’, ‘financiero’, ‘lo’, ‘que’, ‘fue’, ‘destruido’, ‘por’, ‘el’, ‘incendio’, ‘en’, ‘el’, ‘túnel’, ‘</s>’]

2.2.2 Conteo de tokens

[19]:
```
def fill_dict(text_array):
    my_dictionary = {}
    for word in text_array:
        for item in word:
            if item in my_dictionary:
                my_dictionary[item] = my_dictionary[item] + 1
            else:
                my_dictionary[item] = 1
    return my_dictionary
```

La función fill_dict crea un diccionario a partir de un arreglo de strings usando cada palabra como llave. Si la llave ya está, se aumenta su cuenta, sino se le asigna un 1. Esta función se usa para

contar la frecuencia de los tokens, tanto de unigramas como de bigramas.

2.2.3 Creación de matriz de frecuencias

Una vez que se obtuvieron los diccionarios de unigramas y bigramas creamos una matriz cuadrada de tamaño del vocabulario y 2 arreglos usando las llaves de los diccionarios.

[ ]:
```
mat = np.zeros(shape=(len(dataset_dict_unigrams) , len(dataset_dict_unigrams) ))
rows = list(dataset_dict_unigrams.keys())
cols = list(dataset_dict_unigrams.keys())
```

Posteriormente llenamos la matriz con la ocurrencia de los bigramas utilizando la fila y la columna correspondiente, de acuerdo a los datos en el diccionario de bigramas.

[ ]:
```
for x in range(0, len(rows)):
    for y in range(0, len(cols)):
        mat[x,y] = search(rows[x], cols[y], dataset_dict_bigrams)
```

[24]:
```
def search(x,y, bigram_dict):
    dict_key = (x,y)
    return bigram_dict.get(dict_key,0)
```

La función search nos permite buscar un bigrama en el diccionario de bigramas y obtener su frecuencia. En caso de que no se encuentre, devuelve un 0.

Finalmente, creamos un dataframe con que tendrá como columnas y filas las llaves de los diccionarios y cuyos datos estarán representados por la matriz de frecuencias mostrada anteriormente.

[ ]:
```
df = pd.DataFrame(mat,
    index=list(dataset_dict_unigrams.keys()),
    columns=list(dataset_dict_unigrams.keys())
)
df
```

2.2.4 Estimador de Máxima Verosimilitud (Maximum Likelihood Estimation, MLE) y suavizado de Laplace

Para calcular la probabilidad de una palabra dada una palabra previa x, se calcula el conteo de los bigramas $C(xy)$ y se normaliza con todos los bigramas que comparten la primera palabra x, que es lo mismo que los unigramas de x, lo que se obtiene con la siguiente fórmula:

$P\left(y|x\right) = \frac{C(xy)}{C(x)}$

Por otra parte, el suavizado de Laplace o ley de Laplace es una técnica sencilla que consiste en proporcionar un poco del espacio de probabilidades a los eventos no vistos. Para aplicar suavizado de Laplace a bigramas se requiere aumentar el conteo de unigramas con el total de tipos de palabras en el vocabulario V.

Esto es: $P_{Lap}\left(w_2|w_1\right) = \frac{C(w_1 w_2)+1}{C(w_1)+V}$

[16]:
```
def p_x_y_df(xy, df, unigram_dict, laplace = False):
    if(laplace):
        return ( df.loc[xy[0]][xy[1]] + 1) /  ( unigram_dict[xy[0]] +␣
 ↪len(unigram_dict))
    return  df.loc[xy[0]][xy[1]]  /  unigram_dict[xy[0]]
```

La función anterior se encarga del cálculo de MLE. El primer parámetro es un bigrama, el segundo un dataframe que contiene las frecuencias y el tercero un diccionario que contiene el vocabulario. Se incluye un parámetro llamado laplace (predeterminado falso) para utilizar suavizado de Laplace en caso de ser necesario.

2.2.5 Probabilidad de una secuencia

Para calcular la probabilidad de una secuencia se usa la siguiente fórmula: $P_{MLE}(w_1...w_n) \approx \prod_{k=1}^{n} P(w_k|w_{k-1})$

[22]:
```
def calculate_sentence_probability(text, dataf, unigram_dict, laplace = False):
    result = 1
    processed_text = preprocess(text)
    bigrams = list(ngrams(processed_text, 2))
    for i in bigrams:
        result = result*p_x_y_df(i, df, unigram_dict, laplace)
    return result
```

Mediante la función anterior se puede calcular la probabilidad de una secuencia. Recibe como parámetros la secuencia, el dataframe de las frecuencias, , el diccionario que contiene el vocabulario y el parámetro opcional laplace que nos permite utilizar suavizado de Laplace.

2.2.6 Obtener las siguientes n palabras más probables

[ ]:
```
def get_most_likely(word, top_n, laplace = False):
    los_ser = df.loc[word]
    results = {}
    for indx in los_ser.items():
        xy = (word, indx[0])
        results[xy] = p_x_y_df(xy, df, dataset_dict_unigrams, laplace)
    ordered = sorted(results.items(), key=lambda x:x[1], reverse=True)
    return list(map(lambda x: x[0][1], ordered[:top_n]))
```

Utilizando la función anteriormente descrita podemos obtener las siguientes n palabras más probables. El primer parámetro es el número de palabras deseadas y el segundo es opcional e indica si queremos usar suavizado de Laplace.

2.3 Resultados

2.3.1 Obteniendo la probabilidad de una oración

Usando los modelos de probabilidad MLE y MLE con suavizado de Laplace, se calcularán las probabilidades de las siguientes oraciones:

- el parlamento debe enviar un mensaje
- el parlamento debe enviar un consejo
- el abismo entre pobres y ricos
- el abismo entre ricos y pobres
- el abismo de la cantera entre pobres y ricos
- la comisión debe ser totalmente transparente
- la comisión debe ser transparente

Los resultados obtenidos se pueden observar en la siguiente tabla:

Oración	Probabilidad MLE	Probabilidad MLE con suavizado de Laplace
el parlamento debe enviar un mensaje	4.4374617510995097e-13	5.988775375404499e-21
el parlamento debe enviar un consejo	3.3572778663495715e-13	9.358930610679253e-20
el abismo entre pobres y ricos	3.807892655558683e-17	1.6730189594858084e-26
el abismo entre ricos y pobres	8.648730905039033e-15	1.16115482545265e-24
el abismo de la cantera entre pobres y ricos	0	2.2100768814795686e-37
la comisión debe ser totalmente transparente	3.59796250589801e-11	7.49629030518359e-19
la comisión debe ser transparente	2.5521547375169884e-09	4.12560040648121e-15

Podemos observar que las probabilidades obtenidas usando MLE son mayores que las que fueron calculadas usando suavizado de Laplace. Es importante notar que en el caso de "el abismo de la cantera entre pobres y ricos", la probabilidad usando MLE es 0 debido a que el bigrama ('la', 'cantera') no ocurre. Sin embargo, al usar MLE con suavizado de Laplace podemos evitar este problema ya que los bigramas que no ocurrieron en el conjunto de datos al menos tendrán una ocurrencia. Esto nos permite obtener una probabilidad para la oración listada anteriormente. No obstante, es poco probable que ocurra dicha oración.

2.3.2 Predicción de palabras

Usando un modelo de n-gramas también se pueden realizar predicción de palabras, esto es ,dada una palabra inicial mostrar las siguientes palabras más probables de acuerdo con los modelos MLE y MLE con suavizado de Laplace. A continuación se muestran las cinco palabras más probables para cada palabra de una oración dada (las de mayor probabilidad aparecen primero).

Para la secuencia *los tribunales nacionales* obtenemos lo siguiente:

los ['estados', 'países', 'derechos', 'que', 'ciudadanos']

tribunales ['nacionales', 'de', '', 'y', 'en']

nacionales ['', 'de', 'y', 'en', 'que']

Podemos observar que para la palabra *los*, las cinco palabras siguientes más probables son *estados*, *paises*, *derechos*, *que* y *ciudadanos*; luego para la palabra *tribunales* las cinco palabras siguientes más

probables son *nacionales*, *de*, *</s>*, *y*, y *en* y finalmente para la palabra *nacionales*, *</s>*, *de*,*y*,*en* y *que*. Es importante resaltar que el modelo construido nos permite predecir que la siguiente palabra más probable después de *tribunales* será *nacionales* para formar la secuencia *tribunales nacionales*.

Para la palabra *la*, obtenemos lo siguiente:

la [‘comisión’, ‘unión’, ‘política’, ‘sra’, ‘ue’]

Observamos que las cinco palabras siguientes más probables son *comisión*, *unión*, *política*, *sra* y *ue*.

Para el caso de la palabra *parlamento*, tenemos que:

parlamento [‘europeo’, ‘’, ‘y’, ‘en’, ‘que’]

Nótese que las cinco palabras siguientes más probables son *europeo*, *</s>*, *y*, *en* y *que*.

A continuación se realizan 3 ejemplos de predicción utilizando el modelo que se construyó: * Para la secuencia *el en consejo del parlamento* se obtiene:

```
en ['el', 'la', 'los', 'este', 'las']

el ['parlamento', 'consejo', 'sr', 'informe', 'que']

consejo ['de', 'y', '</s>', 'europeo', 'que']

del ['consejo', 'parlamento', 'sr', 'grupo', 'día']

parlamento ['europeo', '</s>', 'y', 'en', 'que']
```

- Para la secuencia *en muchos paises* se obtiene:

  ```
  en ['el', 'la', 'los', 'este', 'las']

  muchos ['de', 'años', 'casos', 'países', 'otros']

  países ['de', 'en', 'que', 'candidatos', '</s>']
  ```

- Para la secuencia *gasto público* se tiene:

  ```
  gasto ['de', 'en', 'público', 'agrícola', '</s>']

  público ['</s>', 'y', 'en', 'de', 'a']
  ```

3 Conclusiones

Los bigramas son muy útiles para la predicción de palabras, ya sea el cálculo de probabilidad de una secuencia o predecir la siguiente palabra en una oración. Sin embargo, son muy dependientes del conjunto de entrenamiento que usemos y son sensibles a encontrarse con probabilidades cero de n-gramas, por lo que es necesario implementar técnicas como el suavizado de Laplace, Backoff, descuento de Good-Turing, etc., para atenuar este problema. A pesar de que en la actualidad existen modelos más complejos como Redes Neuronales Recurrentes o Modelo de Lenguaje Colosal, los n-gramas proporcionan un buen enfoque para tareas de PLN y son fáciles de implementar. Asimismo,

son fundamentales para entender las tareas de modelado de lenguaje y pueden ser usados en gran variedad de aplicaciones.