

Datos del alumno (Nombre y Apellidos): Carlos Andres Giraldo Osorio

Fecha: 02/05/2022

Trabajo: Etiquetado morfosintáctico

Objetivos

Con esta actividad se tratará de que el alumno consiga aplicar un método basado en modelos ocultos de Markov (HMM) para realizar el etiquetado morfosintáctico de una oración.

Descripción

En esta actividad debes implementar en Python un etiquetador morfosintáctico basado en modelos ocultos de Markov (HMM) y realizar el etiquetado morfosintáctico de la oración:

Habla con el enfermo grave de trasplantes.

Implementando también en Python el algoritmo de Viterbi.

Parte 1: Construir el etiquetador morfosintáctico

En esta primera parte de la actividad tienes que implementar en Python el etiquetador morfosintáctico basado en un HMM bigrama a partir de un corpus etiquetado.

Para ello debes utilizar el corpus mia07_t3_tra_Corpus-tagged, que se encuentra disponible en el aula virtual.

El corpus se compone de frases en español etiquetadas con conocimiento sobre las partes de la oración (categorías gramaticales o POS tags). Estas frases etiquetadas han sido extraídas de algunos documentos que forman parte de Wikicorpus, un corpus trilingüe (español, catalán e inglés) compuesto por más de 750 millones de palabras. Wikicorpus fue creado por investigadores de la Universitat Politècnica de Catalunya a partir de documentos de la Wikipedia que fueron anotados con la librería opensource FreeLing.

La tabla 1 muestra en formato de texto plano y sin etiquetar algunos ejemplos de frases que componen el corpus. De hecho, también se indica el identificador del documento del cual han sido extraídas las frases etiquetadas.

La versión anotada la conforma el corpus anotado proporcionado para realizar esta actividad. El formato del fichero de texto que contiene el corpus es el mismo que el utilizado en Wikicorpus. Por lo tanto, cada uno de los documentos de Wikipedia se identifica con el tag XML donde se indica el identificador del documento (id).

Además, cada una de las frases en el documento viene separada por una línea en blanco. La información relativa a cada palabra de la frase se representa en una nueva línea del fichero. Para cada palabra, es decir, en cada línea del fichero, se proporciona —además del token que representa a la propia palabra— su lema, la etiqueta gramatical (POS tag) asociada a la palabra y el sentido de esta.

La figura 1 muestra una captura del corpus anotado, donde se observa la frase «Tristana es una película del director español nacionalizado mexicano Luis Buñuel.» perteneciente al documento de Wikicorpus con identificador 27315 y titulado Tristana.

Si se analizan las anotaciones para la palabra «es», se observa que su lema es «ser», que la categoría gramatical a la que pertenece esa palabra es la identificada por la etiqueta gramatical «VSI3S0» y que el sentido de la palabra es el identificado por el código «01775973175».

También se observa que la palabra «del» en la frase se representa en dos líneas y se anota con dos tokens, el primero «de» y el segundo «el». Esto se debe a que la palabra «del» es la contracción de la preposición «de» y el artículo «el». Por el contrario, el nombre propio «Luis Buñuel», que está formado por dos palabras (el nombre «Luis» y el apellido «Buñuel»), se anota como un único token «luis_buñuel». Además, se observa que el punto final de la frase también viene anotado como un token «.».

Aunque el corpus anotado proporciona más información (ver figura 1), es importante tener en cuenta de que para realizar esta actividad solo será necesario el token y la etiqueta gramatical (POS tag) de cada palabra; es decir, la información contenida en la primera y la tercera cadena de cada línea que representa una palabra en el corpus anotado.

Las etiquetas gramaticales (POS tags) utilizadas para anotar la información morfosintáctica del corpus son las definidas en FreeLing y se basan en EAGLES, una recomendación para la anotación de la mayoría de las lenguas europeas. La definición del conjunto de etiquetas gramaticales (POS tags) utilizadas por FreeLing en el etiquetado de un corpus en español se puede consultar en la web.

Accede al recurso a través del aula virtual o desde la siguiente dirección web: <https://freeling-user-manual.readthedocs.io/en/v4.1/tagsets/tagset-es/>

Las etiquetas gramaticales de EAGLES utilizadas por FreeLing son de longitud variable, donde cada carácter corresponde a una característica morfosintáctica. El primer carácter en la etiqueta es siempre la categoría gramatical o parte de la oración. Esa categoría gramatical determina la longitud de la etiqueta y la interpretación de cada uno del resto de caracteres en la misma.

La definición de la etiqueta para la categoría gramatical «verbo» se muestra en la tabla 2. Entonces, la etiqueta «VSIP3S0», con la que ha sido etiquetada la palabra «es» en la frase que se presentó anteriormente, se interpreta de la siguiente forma: se refiere a un verbo (V) de tipo semiauxiliar (S) en modo indicativo (I) y en tiempo presente (P) para la tercera persona (3) de (número) singular (S). Asimismo, el carácter «0» al final de la etiqueta indica que esta forma verbal no tiene género.

Es importante destacar que para realizar la actividad se deben utilizar las etiquetas con las que se anota el corpus en formato EAGLES; por ejemplo, «VSIP3S0».

Importante: Si se utilizan otras etiquetas la actividad será considerada incorrecta y puntuada con cero puntos.

Para construir el etiquetador morfosintáctico a partir del corpus etiquetado con los datos de entrenamiento, deberás seguir los siguientes pasos:

- Cargar el corpus para extraer la primera y tercera columna de cada registro.
- Calcular las probabilidades que rigen el HMM bigrama, es decir:
 - Calcular las probabilidades de emisión del HMM a partir del corpus etiquetado.
 - Calcular las probabilidades de transición del HMM a partir del corpus etiquetado.

Nota: Presenta en el envío de la actividad la tabla (guardada en formato de hoja de cálculo de Microsoft Excel (.xlsx) o equivalente) con las probabilidades de emisión y las de transición, calculadas para todas las etiquetas y tokens (palabras) que aparecen en el corpus.

Cargar el corpus para extraer la primera y tercera columna de cada registro

En primer lugar se va a cargar el corpus leyendo el archivo y recuperando la información de la *primera* y *tercera* columna de cada registro que contienen el *token* de la palabra y la *etiqueta*, respectivamente.

Estos valores se almacenarán en objetos de la clase `Palabra`.

Esta clase permitirá recuperar el `Token()` y el `Tag()` fácilmente para cada registro.

```
In [1]: class Palabra:
...     """
...     Clase para guardar el token y la etiqueta de una palabra de un corpus
...     """
...
...     def __init__(self, token: str, tag: str):
...         """
...         Constructor de la clase
...         """
...         token : str
...             Token de la palabra
...
...         tag : str
...             Etiqueta de la palabra
...         self._token = token
...         self._tag = tag
...
...     def Token(self):
...         """
...         Método para acceder al token de la palabra
...         """
...         return self._token
...
...     def Tag(self):
...         """
...         Método par acceder a la etiqueta de la palabra
...         """
...         return self._tag
```

El corpus se guardará como una lista que a su vez contiene una serie de listas de objetos del tipo `Palabra`. Cada una de las listas de objetos del tipo `Palabra` guarda una oración.

```
In [2]: archivo = open('mia07_t3_tra_Corpus-tagged.txt', "r")

corpus = list()
oracion_actual = list()

for entrada in archivo.readlines():
    entrada = entrada.split()
    if len(entrada) == 0:
```

```

# Puede ser La primera oración del documento
# O que termina La oración
if len(oracion_actual) > 0:
    # Fin de La oración
    corpus.append(oracion_actual)
    oracion_actual = list()
    continue

elif entrada[0] == '<doc':
    # Inicio de documento. No se hace nada
    continue

elif entrada[0] == '</doc>':
    # Fin del documento. No se hace nada
    continue

oracion_actual.append(Palabra(token=entrada[0], tag=entrada[2]))

archivo.close()

```

In [3]: `# corpus`

El siguiente código te permite imprimir el corpus:

In [4]:

```

# for oracion in corpus:
#     for palabra in oracion:
#         print(palabra.Token(), palabra.Tag())
#     print('')

```

Calcular las probabilidades que rigen el HMM bigrama

Una vez se dispone del `corpus` correctamente cargado se creará un objeto, `hmmbigrama` de la clase `HMMBigrama`.

`hmmbigrama` permitirá hacer el cálculo de las tablas de probabilidades de transición y de emisión.

In [5]:

```

#Se usa pandas para crear las tablas.
import pandas as pd

class HMMBigrama:
    """
    Clase para obtener las matrices de probabilidad HMM Bigrama a partir de un corpus
    """

    def __init__(self, corpus: [[Palabra]]):
        """
        Constructor de la clase para calcular el Modelo Oculto de Markov Bigrama
        """
        self._corpus = corpus
        self._estados = dict()
        self._tokens = dict()
        self._q0 = 'q0'
        self._qF = 'qF'

        self._prob_trans = pd.DataFrame()
        self._prob_obs = pd.DataFrame()

    def Corpus(self):
        return self._corpus.copy()

    def EstadoInicial(self):
        return self._q0

    def EstadoFinal(self):
        return self._qF

    def _ProcesarCorpus(self):
        """
        Método para contar el número de ocurrencias de estados y tokens
        """
        for oracion in self._corpus:
            for palabra in oracion:

                # Se recorren todas las palabras de todas las oraciones del corpus recuperando las etiquetas (estados)
                estado = palabra.Tag()
                estados = self._estados
                estados[estado] = estados[estado] + 1 if estado in estados else 1

                # Se recorren todas las palabras de todas las oraciones del corpus recuperando los tokens

```

```

        token = palabra.Token()
        tokens = self._tokens
        tokens[token] = tokens[token] + 1 if token in tokens else 1

def Estados(self, incluir_inicial: bool = False, incluir_final: bool = False):
    """
    Devuelve los estados del bigrama en base al corpus proporcionado al constructor

    incluir_inicial : bool (False)
        Flag para indicar si se quiere recuperar el estado inicial

    incluir_final : bool (False)
        Flag para indicar si se quiere recuperar el estado final

    return
        Diccionario de estados con el número de ocurrencias de cada estado en el corpus
    """

    if len(self._estados) == 0:
        self._ProcesarCorpus()

    copia_estados = dict()
    if incluir_inicial:
        # Hay tantos estados como oraciones en el corpus
        copia_estados[self._q0] = len(self._corpus)

    copia_estados.update(self._estados)

    if incluir_final:
        # Hay tantos estados como oraciones en el corpus
        copia_estados[self._qF] = len(self._corpus)

    return copia_estados

def Tokens(self):
    """
    Devuelve los tokens del bigrama en base al corpus proporcionado al constructor

    return
        Diccionario de tokens con el número de ocurrencias de cada token en el corpus
    """

    if len(self._tokens) == 0:
        self._ProcesarCorpus()

    return self._tokens.copy()

def ProbabilidadesDeTransicion(self):
    """
    Método para calcular las probabilidades de transición bigrama
    a partir del corpus proporcionado a la clase
    """

    # Si ya se ha calculado se devuelve
    if len(self._prob_trans) != 0:
        return self._prob_trans.copy()

    """
    En esta parte del código se calcula el número de
    transiciones bigrama, es decir, en el diccionario
    'contador_transiciones' se almacenarán los contadores
    de las transiciones t-1 -> t

    Las claves del diccionario serán los estados de partida
    mientras que los valores de cada clave serán los estados
    de destino y el número de veces que transitan a cada estado
    """

    q0 = self._q0
    qF = self._qF
    contador_transiciones = {q0: dict()}

    for oracion in self._corpus:
        # Contador de transición q0 a estado q1
        q1 = oracion[0].Tag()
        if q1 not in contador_transiciones[q0]:
            contador_transiciones[q0][q1] = 0
        contador_transiciones[q0][q1] += 1

        # Contador de transiciones entre palabras de la oración
        for it in range(0, len(oracion) - 1): # Se hace la siguiente lógica para cada token de cada oración:

```

```

q_0 = oracion[it].Tag() # Se toma la etiqueta del token actual
q_1 = oracion[it+1].Tag() # Se toma la etiqueta del token siguiente

# Si la etiqueta actual no existe en el diccionario que guarda la cuenta de transiciones, se debe crear
if q_0 not in contador_transiciones:
    contador_transiciones[q_0]=dict()

# Si la etiqueta siguiente no existe en el diccionario que guarda la cuenta de las transiciones de la etiqueta actual
if q_1 not in contador_transiciones[q_0]:
    contador_transiciones[q_0][q_1] = 0

# Se suma 1 a la cuenta de las transiciones de la etiqueta actual a la siguiente
contador_transiciones[q_0][q_1] += 1

# Contador de transición qF_1 a qF
qF_1 = oracion[-1].Tag()

if qF_1 not in contador_transiciones:
    contador_transiciones[qF_1] = dict()
if qF not in contador_transiciones[qF_1]:
    contador_transiciones[qF_1][qF] = 0

contador_transiciones[qF_1][qF] += 1

'''
Cálculo de la tabla de probabilidades de transición.

Se calculan ahora las probabilidades de transición
siguiendo la relación:  $P(T|T-1) = C(T-1, T) / C(T-1)$ .

En 'contador_transiciones' se han acumulado las coincidencias  $C(T-1, T)$ 
y en 'estados' se tiene disponible  $C(T-1)$  por lo que es posible
calcular la tabla de probabilidades de transiciones con estos elementos.
'''

tags_estados_iniciales = list(
    self.Estados(incluir_inicial=True).keys())
tags_estados_finales = list(self.Estados(incluir_final=True).keys())
estados_totales = self.Estados(
    incluir_inicial=True, incluir_final=True)

prob_trans = {qt_1: {qt: 0 for qt in tags_estados_finales}
               for qt_1 in tags_estados_iniciales}
for qt_1 in tags_estados_iniciales:
    for qt in tags_estados_finales:
        prob = 0
        if qt_1 in contador_transiciones and qt in contador_transiciones[qt_1]:
            # Para hallar la probabilidad de transición de una etiqueta dada la anterior
            # Se calcula la división entre el contador de las transiciones de la etiqueta anterior a la actual,
            # y la cuenta de las veces que aparece la etiqueta anterior
            prob = contador_transiciones[qt_1][qt]/estados_totales[qt_1]

        prob_trans[qt_1][qt] = prob

self._prob_trans = pd.DataFrame.from_dict(prob_trans, orient='index')

return self._prob_trans.copy()

def ProbabilidadesDeEmision(self):
    '''
    Método para calcular las probabilidades de emisión
    a partir del corpus proporcionado a la clase
    '''

    if len(self._prob_obs) != 0:
        return self._prob_obs.copy()

    '''
    En esta parte del código se calculan el número de
    ocurrencias de la palabra  $W_i$  para la etiqueta  $T_i$ 
    '''

    estados = self.Estados()
    contador_observaciones = {key: dict() for key in estados.keys()}

    for oracion in self._corpus:
        for palabra in oracion:
            token = palabra.Token()
            etiqueta = palabra.Tag()
            # Si la palabra no existe en el diccionario que guarda la cuenta de cada palabra para una etiqueta, se debe crear
            if token not in contador_observaciones[etiqueta]:
                contador_observaciones[etiqueta][token] = 0
            contador_observaciones[etiqueta][token] += 1

```

```

'''
Cálculo de la tabla de probabilidades de emisión.

Se calculan ahora las probabilidades de emisión
siguiendo la relación:  $P(W_i|T_i) = C(T_i, W_i) / C(T_i)$ .

En 'contador_observaciones' se han acumulado la coincidencias  $C(T_i, W_i)$ 
y en 'estados' se tiene disponible  $C(T_i)$  por lo que es posible
calcular la tabla de probabilidad de emisión con estos elementos.
'''
tokens = self.Tokens()
prob_obs = {Ti: {Wi: 0 for Wi in tokens} for Ti in estados}
for Ti in estados:
    for Wi in tokens:
        prob = 0
        if Ti in contador_observaciones and Wi in contador_observaciones[Ti]:
            prob = contador_observaciones[Ti][Wi] / estados[Ti]
        prob_obs[Ti][Wi] = prob

self._prob_obs = pd.DataFrame.from_dict(prob_obs, orient='index')

return self._prob_obs

```

El siguiente código te permite crear el HMM Bigrama y obtener información relevante:

```
In [6]: hmbbigrama = HMMBigrama(corpus)
```

```
In [7]: # hmbbigrama.Tokens()
```

```
In [8]: len(hmbbigrama.Tokens())
```

```
Out[8]: 1501
```

```
In [9]: # hmbbigrama.Estados()
```

```
In [10]: len(hmbbigrama.Estados())
```

```
Out[10]: 134
```

El método ProbabilidadesDeTransición() de la clase HMMBigrama devuelve la tabla de probabilidades de transición.

```
In [11]: def non_zero_green(val):
'''
Función para resaltar en verde las probabilidades que no sean 0
'''
return 'background-color: Aquamarine' if val > 0 else ''
```

```
In [12]: prob_transicion = hmbbigrama.ProbabilidadesDeTransicion()
#prob_transicion.style.applymap(non_zero_green)
```

```
In [13]: prob_transicion.to_excel('mia07_t3_tra_resultados_trans.xlsx', sheet_name='prob_trans')
```

El método ProbabilidadesDeEmisión() de la clase HMMBigrama devuelve la tabla de probabilidades de emisión.

```
In [14]: prob_emision = hmbbigrama.ProbabilidadesDeEmision()
#prob_emision.style.applymap(non_zero_green)
```

```
In [15]: prob_emision.to_excel('mia07_t3_tra_resultados_emision.xlsx', sheet_name='prob_emision')
```

Parte 2: Etiquetar morfosintácticamente una oración

En esta segunda parte de la actividad tienes que implementar en Python un programa que permita calcular la mejor secuencia de etiquetas para una oración, dicho de otro modo, realizar el etiquetado morfosintáctico de la oración: «Habla con el enfermo grave de trasplantes. ».

Para ello debes utilizar el etiquetador que has construido en la parte 1 de esta actividad, es decir las tablas de probabilidades calculadas, y aplicar el algoritmo de Viterbi.

Para aplicar el algoritmo de Viterbi, se deben seguir los siguientes pasos:

- Calcular la matriz de probabilidades de la ruta se Viterbi (matriz con los valores de Viterbi) donde se representen claramente las observaciones y los estados de la máquina de estados finitos. Calcula el valor de Viterbi para cada celda de la matriz e indica claramente los valores obtenidos. Nota: Para simplificar, puedes eliminar todos aquellos estados asociados a etiquetas que no aparezcan en el posible análisis de la oración y sólo quedarte con los estados relevantes. Además, debes tener en cuenta la transición al estado final representado por el punto al final de la oración a analizar.
- Obtener la ruta con máxima probabilidad, es decir, traza la ruta inversa para obtener la mejor secuencia de etiquetas.
- Mostrar la oración etiquetada. Debes indicar claramente el resultado obtenido del etiquetado morfosintáctico de la oración estudiada.

Nota: Presenta en el envío de la actividad la tabla (guardada en formato de hoja de cálculo de Microsoft Excel (.xlsx) o equivalente) con la matriz de probabilidades de la ruta Viterbi para el etiquetado morfosintáctico de la oración «Habla con el enfermo grave de trasplantes. ».

Calcular la matriz de probabilidades de la ruta de Viterbi

La clase `Viterbi` permitirá realizar el cálculo de la matriz de probabilidades de la ruta de Viterbi y la posterior decodificación de la secuencia óptima de etiquetado para una oración a analizar.

El etiquetado morfosintáctico creado en la Parte 1, es decir el objeto `hmmbigrama` de la clase `HMMBigrama`, será proporcionado al objeto `viterbi` de la clase `Viterbi` para poder aplicar el Algoritmo de Viterbi.

El cálculo de los valores de Viterbi se realiza en el método `Probabilidades()` de la clase `Viterbi`.

Obtener la ruta con máxima probabilidad

El método `DecodificacionSecuenciaOptima()` de la clase `Viterbi` permite obtener la secuencia de etiquetas más probables para la oración a analizar.

```
In [16]: class Viterbi:
    """
    Algoritmo de Viterbi para obtener las mejores
    etiquetas de las palabras de una oración
    """

    def __init__(self, hmmbigrama: HMMBigrama, oracion: str):
        self._hmmbigrama = hmmbigrama
        self._oracion = oracion

        self._estados_relevantes = None
        self._prob_viterbi = pd.DataFrame()
        self._estado_max_anterior = None

    def _CalculoEstadosRelevantes(self):
        self._estados_relevantes = set()
        for palabra_analizar in [x.lower() for x in self._oracion.split()]:
            # Búsqueda de estados
            for oracion in self._hmmbigrama.Corpus():
                for palabra_corpus in oracion:
                    if palabra_corpus.Token() == palabra_analizar:
                        self._estados_relevantes.add(palabra_corpus.Tag())

    def Probabilidades(self):
        #if len(self._prob_viterbi) != 0:
        #    return self._prob_viterbi.copy()

        if not self._estados_relevantes:
            self._CalculoEstadosRelevantes()

        estados_relevantes = self._estados_relevantes

        """
        Matriz en la que se guardan los valores de Viterbi
        """
        matriz_viterbi = {q: dict() for q in estados_relevantes}

        """
        Matriz asociada a la matriz de Viterbi en la que se almacena
        el estado de origen que maximiza cada probabilidad
        """
        self._estado_max_anterior = {q: dict() for q in estados_relevantes}

        q0 = self._hmmbigrama.EstadoInicial()
        prob_trans = self._hmmbigrama.ProbabilidadesDeTransicion()
        prob_obs = self._hmmbigrama.ProbabilidadesDeEmision()
```

```
token_anterior = None
for token in [x.lower() for x in self._oracion.split()]:
    for qDestino in estados_relevantes:

        prob_max = 0
        if not token_anterior:
            # Estado q0
            prob_max = prob_trans[qDestino][q0]
        else:
            # Resto de estados
            for qOrigen in estados_relevantes:
                # Se calcula la probabilidad del token anterior con cada etiqueta por
                # La probabilidad de transición de la etiqueta actual dada la anterior
                # Solo se toman estos 2 factores porque más adelante se multiplica por la probabilidad de emisión
                prob_qOrigen = matriz_viterbi[qOrigen][token_anterior] * prob_trans[qDestino][qOrigen]

            if prob_qOrigen > prob_max:
                # Se guarda el valor máximo de probabilidad
                prob_max = prob_qOrigen
                # Se guarda la etiqueta anterior que permitió maximizar la probabilidad
                self._estado_max_anterior[qDestino][token] = qOrigen

        matriz_viterbi[qDestino][token] = prob_max * prob_obs[token][qDestino]

    token_anterior = token

self._prob_viterbi = pd.DataFrame.from_dict(matriz_viterbi, orient='index')

return self._prob_viterbi.copy()

def DecodificacionSecuenciaOptima(self):
    # Decodificación de la secuencia óptima
    oracion_invertida = [x.lower() for x in self._oracion.split()]
    oracion_invertida.reverse()

    prob_viterbi = self.Probabilidades()

    oracion_etiquetada = []
    # Se busca la probabilidad máxima de Viterbi asociada a la última palabra de la oración
    palabra = oracion_invertida[0]
    etiqueta = prob_viterbi[palabra].idxmax()
    oracion_etiquetada.append({'token': palabra, 'tag': etiqueta, 'prob': prob_viterbi[palabra].max()})

    # Ahora se usa la tabla auxiliar de Viterbi que contiene
    # el estado de origen que maximiza cada probabilidad Viterbi
    palabra_anterior = palabra
    for palabra in oracion_invertida[1:]:
        # Buscamos la etiqueta que permitió maximizar la probabilidad de la etiqueta del token anterior
        etiqueta = self._estado_max_anterior[etiqueta][palabra_anterior]
        # Se guarda la palabra para una próxima iteración
        palabra_anterior = palabra
        # Se guarda el resultado de la etiqueta seleccionada y su probabilidad para la palabra actual
        oracion_etiquetada.append({'token': palabra, 'tag': etiqueta, 'prob': prob_viterbi[palabra][etiqueta]})

    # Se recupera el orden de la oración con las palabras ya etiquetadas
    oracion_etiquetada.reverse()

    return oracion_etiquetada
```

El siguiente código te permite realizar el análisis de la oración: "Habla con el enfermo grave de trasplantes."

```
In [17]: viterbi = Viterbi(hmmbigrama=hmmbigrama, oracion='Habla con el enfermo grave de trasplantes .')
```

El siguiente código te permite mostrar la matriz de probabilidades de la ruta de Viterbi (solo se presentan aquellas etiquetas que tienen algún valor no nulo para alguna de las palabras de la oración analizada).

```
In [18]: matriz_prob_viterbi = viterbi.Probabilidades()
matriz_prob_viterbi.style.applymap(non_zero_green)
```

Out[18]:

	habla	con	el	enfermo	grave	de	trasplantes	.
SP500	0.000000	0.000020	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
AQ0CS0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
DA0MS0	0.000000	0.000000	0.000003	0.000000	0.000000	0.000000	0.000000	0.000000

	habla	con	el	enfermo	grave	de	trasplantes	.
NCMP000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
NCFS000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
VMIP350	0.001294	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
NCMS000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
AQ0MS0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
Fp	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

```
In [19]: matriz_prob_viterbi.to_excel('mia07_t3_tra_resultados_viterbi.xlsx', sheet_name='viterbi')
```

El siguiente código te permite mostrar la ruta de Viterbi con máxima probabilidad

```
In [20]: oracion_etiquetada = viterbi.DecodificacionSecuenciaOptima()
```

```
In [21]: oracion_etiquetada
```

```
Out[21]: [{'token': 'habla', 'tag': 'VMIP350', 'prob': 0.0012941074971961003},
{'token': 'con', 'tag': 'SPS00', 'prob': 1.9880631335173152e-05},
{'token': 'el', 'tag': 'DA0MS0', 'prob': 2.779888150251198e-06},
{'token': 'enfermo', 'tag': 'NCMS000', 'prob': 2.8278183745351773e-08},
{'token': 'grave', 'tag': 'AQ0CS0', 'prob': 6.385259012197449e-11},
{'token': 'de', 'tag': 'SPS00', 'prob': 5.440257583147429e-12},
{'token': 'trasplantes', 'tag': 'NCMP000', 'prob': 4.107075865754602e-15},
{'token': '.', 'tag': 'Fp', 'prob': 3.529518322132861e-16}]
```

Mostrar la oración etiquetada

El siguiente código te permite mostrar la oración etiquetada

```
In [22]: for palabra in oracion_etiquetada:
print('{ } / {}'.format(palabra['token'], palabra['tag']))
```

habla / VMIP350
con / SPS00
el / DA0MS0
enfermo / NCMS000
grave / AQ0CS0
de / SPS00
trasplantes / NCMP000
. / Fp

Parte 3: Analizar el etiquetador morfosintáctico

Una vez hayas creado el etiquetador morfosintáctico y lo hayas utilizado para etiquetar la oración «Habla con el enfermo grave de trasplantes.», reflexiona sobre los resultados obtenidos, interprétalos y analiza el rendimiento del etiquetador creado y sus limitaciones. Para ello responde de forma razonada a las siguientes preguntas:

- ¿Es correcto el etiquetado morfosintáctico que has obtenido? Indica por qué.

A continuación presentaré la descripción del formato de etiquetas EAGLES de la frase etiquetada:

habla / VMIP350 : Verbo Principal Imperativo Presente Tercera Persona Singular
con / SPS00 : Adposición Preposición Simple
el / DA0MS0 : Determinante Artículo Masculino singular
enfermo / NCMS000 : Nombre Común Masculino Singular
grave / AQ0CS0 : Adjetivo Calificativo Común Singular
de / SPS00 : Adposición Preposición Simple
trasplantes / NCMP000 : Nombre Común Masculino Plural
. / Fp : Puntuación Punto

El etiquetado morfosintáctico obtenido es correcto debido a que si detallamos las etiquetas obtenidas podemos ver que el etiquetador no falló en ninguna de las etiquetas de esa frase en específico.

- Indica el resultado de etiquetar la oración «El enfermo grave habla de trasplantes.» utilizando el etiquetador morfosintáctico. ¿Es correcto el etiquetado morfosintáctico que has obtenido? Indica por qué.

```
In [23]: viterbi2 = Viterbi(hmmbigrama=hmmbigrama, oracion='El enfermo grave habla de trasplantes .')
matriz_prob_viterbi2 = viterbi2.Probabilidades()
oracion_etiquetada2 = viterbi2.DecodificacionSecuenciaOptima()
```

```
In [24]: oracion_etiquetada2
```

```
Out[24]: [{'token': 'el', 'tag': 'DA0MS0', 'prob': 0.054121718640698056},
{'token': 'enfermo', 'tag': 'NCMS000', 'prob': 0.0005505487349185588},
{'token': 'grave', 'tag': 'AQ0CS0', 'prob': 1.2431478283574262e-06},
{'token': 'habla', 'tag': 'NCFS000', 'prob': 6.283919670208897e-10},
{'token': 'de', 'tag': 'SPS00', 'prob': 7.851971394520127e-11},
{'token': 'trasplantes', 'tag': 'NCMP000', 'prob': 5.927778550950864e-14},
{'token': '.', 'tag': 'Fp', 'prob': 5.094184692223398e-15}]
```

```
In [25]: for palabra in oracion_etiquetada2:
        print('{} / {}'.format(palabra['token'], palabra['tag']))
```

```
el / DA0MS0
enfermo / NCMS000
grave / AQ0CS0
habla / NCFS000
de / SPS00
trasplantes / NCMP000
. / Fp
```

A continuación presentaré la descripción del formato de etiquetas EAGLES de la frase etiquetada:

```
el / DA0MS0 : Determinante Artículo Masculino singular
enfermo / NCMS000 : Nombre Común Masculino Singular
grave / AQ0CS0 : Adjetivo Calificativo Común Singular
habla / NCFS000 : Nombre Común Femenino Singular
de / SPS00 : Adposición Preposición Simple
trasplantes / NCMP000 : Nombre Común Masculino Plural
. / Fp : Puntuación Punto
```

En este caso el etiquetador morfosintáctico ha fallado en una de las etiquetas, "habla" en realidad es un Verbo Principal Imperativo Presente Tercera Persona Singular. Aunque el etiquetado no fue perfecto, acertó en la mayoría de las palabras, aun así, se puede decir que el etiquetado fue incorrecto.

- ¿Cuáles son las limitaciones del analizador morfosintáctico que has creado?

Las oraciones son tratadas como simples concatenaciones de palabras con un tag correspondiente. El etiquetador construido solo se basa en la probabilidad de transición entre dos estados y la probabilidad de emisión de la palabra, pero deja por fuera aspectos importantes como la función específica de la palabra en la oración y como se relaciona con las otras palabras de la oración. Además, es un método solamente estadístico del corpus usado, con el uso de algún algoritmo de aprendizaje automático que tenga en cuenta diferentes características de las palabras dentro de la oración, podría mejorar los resultados.

- ¿Qué posibles mejoras se podrían aplicar para mejorar el rendimiento del etiquetador morfosintáctico creado?

Como se explicó en la pregunta anterior, ampliando las características tenidas en cuenta para el etiquetado, como por ejemplo, la función específica de la palabra en la oración y como se relaciona con las otras, podría mejorar los resultados. Además, usaría otro algoritmo para la predicción de la etiqueta como por ejemplo, redes neuronales que reciban como entrada diferentes características de las palabras de la oración.