

Universidad Internacional de La Rioja (UNIR) - Máster Universitario en Inteligencia Artificial - Procesamiento del Lenguaje Natural

Datos del alumno (Nombre y Apellidos): German Arley Portilla gonzalez

Fecha: 4-05-2022

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 «VSIP3S0» 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/> (<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 [85]: 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 [88]: ##Se imprime el corpus
for oracion in corpus:
    for palabra in oracion:
        print(palabra.Token(), palabra.Tag())
```

```
Tristana NP00000
es VSIP3S0
una DI0FS0
pel cula NCFS000
de SPS00
el DA0MS0
director NCMS000
espa ol AQ0MS0
nacionalizado VMP00SM
mexicano AQ0MS0
Luis_Bu uel NP00000
. Fp
Est  VAIP3S0
basada VMP00SF
en SPS00
la DA0FS0
novela NCFS000
de SPS00
el DA0MS0
. AQ0MS0
```

Calcular las probabilidades que rigen el HMM bigrama

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

`hmmbigrama` permitir  hacer el c culo de las tablas de probabilidades de transici n y de emisi n.

```

In [89]: #Se usa pandas para crear Las tablas.
# Las probabilidades de transmision y emision se generan a partir de lo visto en

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
                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
                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
    ...

    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 constr

    return
        Diccionario de tokens con el número de ocurrencias de cada token en e
    ...

    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

```



```

...
# El metodo se encuentra incompleto para esta practica, dado por el for a
# sobre la oracion donde se almacena cada una de las transiciones bigram

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):

        #####
        ##### Aquí debes incluir tu código #####
        #####

        if oracion[it].Tag() not in contador_transiciones:
            contador_transiciones[oracion[it].Tag()] = dict()
        if oracion[it+1].Tag() not in contador_transiciones[oracion[it].Tag()]:
            contador_transiciones[oracion[it].Tag()][oracion[it+1].Tag()] = 0

        contador_transiciones[oracion[it].Tag()][oracion[it+1].Tag()] += 1
    # El apartado de codigo corresponde al conteo de las transiciones o t
    # de transicion con las cordnadas de desplazamiento q0 y q1 respecti
    # se generan dos condiciones secuenciales que obedecen a la verificac
    # dado sobre la oracion con el iterador it se desplaza en las psocia
    # para la segunda condicion se contempla el iterador para el paso sig
    # cada vez que se cumple la transicion dispuesta por t-1 hacia t se c
    # de la probabilidad, en la cual se hace necesario el total que perm

    # 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 la 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:
            #####
            ##### Aquí debes incluir tu código #####
            #####
            probabilidad_trans = (contador_transiciones[qt_1][qt]) / (estados_totales[qt_1])
            prob = probabilidad_trans

# Las probabilidades se daran de forma independiente para cada una de las transiciones
# estas probabilidades se calculan mediante la relacion vista en clase, y dividida por el total
# cada probabilidad es igual al contador de transiciones para cada t-1 que se divide por el total
# que se representa en el algormimo mediante qt, esto para cada una de las transiciones
# la division del valor con el estado total suministrado en el apartado anterior
# t-1 que en el algormimo se describe como qt_1, luego de generados cada una de las probabilidades
# puesto que para cada transicion existe un valor de probabilidad calculada,
# iterador del calculo anterior.

        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 Wi para la etiqueta Ti
    """

    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()
            if token not in contador_observaciones[etiqueta]:
                #####
                ##### Aquí debes incluir tu código #####
                #####

            contador_observaciones[etiqueta][token] = 0

```

```

# para las probabilidades de emision se ejecuta un proceso similar al anterior
# que depende directamente del for de recorrido con un iterador sobre la oracion
# se presenta la condicion cuando se cumple la asignacion estaria definida por
# donde al no determinar la condicion se asignaria un cero puesto que la condicion
# de esta manera se va sumando en cada una de las coordenadas para el contador
# las probabilidades de las emisiones, las coordenadas del contador de observaciones
# de esta manera se va desplazando entre la lista de listas con acumulacion en

```

```

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 las 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]:

```

```

            #####

```

```

            ##### Aquí debes incluir tu código #####

```

```

            #####

```

```

            probabilidad_emisio = contador_observaciones[Ti][Wi] / estados[Ti]

```

```

            prob = probabilidad_emisio

```

```

# Cada una de las probabilidades de emision o de observacion como se trabajaron en la
# transmision pero esta vez el cambio radica en la variable que estaria para cada
# en coordenada etiqueta, token que para este recorrido estaria por la palabra
# da un valor de probabilidad para cada lista de lista comprendida en la matriz
# con la coordenada etiqueta Ti, que seria solo el valor de una coordenada de la
# se lleva a la matrix de probabilidades de emision u observacion dada por la

```

```

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 [90]: hmmbigrama = HMMBigrama(corpus)

```



```
In [95]: 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 [96]: prob_transicion = hmmbigrama.ProbabilidadesDeTransicion()
prob_transicion.style.applymap(non_zero_green)

# se resaltan las probabilidades diferentes de cero con el color verde
```

Out[96]:

	NP00000	VSIP3S0	DI0FS0	NCFS000	SPS00	DA0MS0	NCMS000	AQ0MS0	VI
q0	0.196532	0.005780	0.017341	0.000000	0.277457	0.057803	0.005780	0.000000	(
NP00000	0.000000	0.021875	0.000000	0.000000	0.078125	0.003125	0.009375	0.000000	(
VSIP3S0	0.000000	0.000000	0.161290	0.000000	0.064516	0.064516	0.000000	0.000000	(
DI0FS0	0.000000	0.000000	0.000000	0.739130	0.065217	0.000000	0.000000	0.000000	(
NCFS000	0.025830	0.011070	0.000000	0.000000	0.361624	0.000000	0.003690	0.003690	(
SPS00	0.149341	0.000000	0.019034	0.058565	0.001464	0.149341	0.045388	0.001464	(
DA0MS0	0.019108	0.000000	0.000000	0.019108	0.012739	0.000000	0.694268	0.019108	(
NCMS000	0.040293	0.007326	0.000000	0.000000	0.322344	0.007326	0.010989	0.040293	(
AQ0MS0	0.048780	0.000000	0.000000	0.000000	0.341463	0.000000	0.219512	0.000000	(
VMP00SM	0.000000	0.000000	0.000000	0.000000	0.421053	0.026316	0.039474	0.026316	(
Fn	0.000000	0.000000	0.000000	0.000000	0.005618	0.000000	0.000000	0.000000	(

```
In [97]: prob_transicion.to_excel('mia07_t3_tra_resultados_trans.xlsx', sheet_name='prob_t
# Se genera el excel con los resultados para cada una de las tranmisiones genera
# como se observa la mayoría de los resultados son cero lo que indica que las inc
# mayoría de los casos pero un buen número presenta resultados relevantes en el co
```

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

```
In [98]: prob_emision = hmmbigrama.ProbabilidadesDeEmision()
prob_emision.style.applymap(non_zero_green)
# Las probabilidades de observacion o de emision apartir del algoritmo implementado
# no es posible precisar los cambios o probabilidades de las etiquetas ti en la p
# calculos.
```

Out[98]:

	Tristana	es	una	película	de	el	director	español	na
NP00000	0.015625	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
VSIP3S0	0.000000	0.903226	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
DI0FS0	0.000000	0.000000	0.847826	0.000000	0.000000	0.000000	0.000000	0.000000	
NCFS000	0.000000	0.000000	0.000000	0.099631	0.000000	0.000000	0.000000	0.000000	
SPS00	0.000000	0.000000	0.000000	0.000000	0.345534	0.000000	0.000000	0.000000	
DA0MS0	0.000000	0.000000	0.000000	0.000000	0.000000	0.936306	0.000000	0.000000	
NCMS000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.010989	0.000000	
AQ0MS0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.024390	
VMP00SM	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 [99]: 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 [100]: 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:

    #####
    ##### Aquí debes incluir tu código #####
    #####
    probabilidad_or = prob_trans[qDestino][qOrigen]
    prob_qOrigen = probabilidad_or

# Para el calculo de la matriz de Viterbi es necesario usar el estado que max
# luego de determinada la probabilidad de maxima para cada coordenada de las
# un for de desplazamiento en la matrix de probabilidades de tranmision dentro
# el metodo de probabilidades mediante self._estados_relevantes cada vez que
# toman las rprobabilidades maximas de transmision.

    if prob_qOrigen > prob_max:

        #####
        ##### Aquí debes incluir tu código #####
        #####
        probabilidad_maxima = prob_qOrigen
        prob_max = probabilidad_maxima

# Luego de obtener el despalamiento por el for de los estados relevantes con
# es mayor a la rpbabilidad maxima esperada para cada captura y se almacena en
# se percibe en el codigo que solo se toman las probabilidades maximas puesto
# de las mejores probabilidades, que posteriormente se almacenan en la matrix
# emision en las coordenadas token, y qDestino multiplicado por la probabilidad

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 probablidad máxima de Viterbi asociada a la última palabra
    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:]:

```

```
#####
```

```

##### Aquí debes incluir tu código #####
#####
etiq = prob_viterbi[palabra].idxmax()
oracion_etiquetada.append({'token': palabra, 'tag': etiq, 'prob': prob})
# en este apartado de código luego de obtener cada una de las etiquetas asociadas
# se genera la matriz con los datos relevantes del etiquetado monosintáctico
# máxima hallada a partir de la probabilidad de transmisión y la de emisión como
# se encuentra en la fase final, por último se genera un ordenamiento de los

# 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 [101]: viterbi = Viterbi(hmmbigrama=hmmbigrama, oracion='Habla con el enfermo grave de t
viterbi2 = Viterbi(hmmbigrama=hmmbigrama, oracion='Habla con el enfermo grave de

```

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 [102]: matriz_prob_viterbi = viterbi.Probabilidades()
matriz_prob_viterbi.style.applymap(non_zero_green)

```

Out[102]:

	habla	con	el	enfermo	grave	de	trasplantes	.
AQ0MS0	0.000000	0.000000	0.000000	0.000983	0.000000	0.000000	0.000000	0.000000
AQ0CS0	0.000000	0.000000	0.000000	0.000000	0.002730	0.000000	0.000000	0.000000
NCMS000	0.000000	0.000000	0.000000	0.010172	0.000000	0.000000	0.000000	0.000000
NCMP000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000755	0.000000
NCFS000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
VMIP3S0	0.001294	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
SPS00	0.000000	0.019590	0.000000	0.000000	0.000000	0.124953	0.000000	0.000000
Fp	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.095941
DA0MS0	0.000000	0.000000	0.139829	0.000000	0.000000	0.000000	0.000000	0.000000

```
In [103]: matriz_prob_viterbi2 = viterbi2.Probabilidades()
matriz_prob_viterbi2.style.applymap(non_zero_green)

## Se prueba la frase con la variacion de incluir o no el punto, para verificar l
# etiquetado y ademas se le calcula los resultados de la matrix de viterbi, para
# punto aunque no es relevante desde mi criterio, puesto que no hace parte de las
```

Out[103]:

	habla	con	el	enfermo	grave	de	trasplantes
AQ0MS0	0.000000	0.000000	0.000000	0.000983	0.000000	0.000000	0.000000
AQ0CS0	0.000000	0.000000	0.000000	0.000000	0.002730	0.000000	0.000000
NCMS000	0.000000	0.000000	0.000000	0.010172	0.000000	0.000000	0.000000
NCMP000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000755
NCFS000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
VMIP3S0	0.001294	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
SPS00	0.000000	0.019590	0.000000	0.000000	0.000000	0.124953	0.000000
DA0MS0	0.000000	0.000000	0.139829	0.000000	0.000000	0.000000	0.000000

```
In [104]: matriz_prob_viterbi.to_excel('mia07_t3_tra_resultados_viterbi.xlsx', sheet_name='')
```

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

```
In [105]: oracion_etiquetada = viterbi.DecodificacionSecuenciaOptima()
```

```
In [106]: oracion_etiquetada
```

```
Out[106]: [{'token': 'habla', 'tag': 'VMIP3S0', 'prob': 0.0012941074971961003},
{'token': 'con', 'tag': 'SPS00', 'prob': 0.019590151977654475},
{'token': 'el', 'tag': 'DA0MS0', 'prob': 0.1398289673695107},
{'token': 'enfermo', 'tag': 'NCMS000', 'prob': 0.010172417815729917},
{'token': 'grave', 'tag': 'AQ0CS0', 'prob': 0.002729616337259263},
{'token': 'de', 'tag': 'SPS00', 'prob': 0.12495340180341774},
{'token': 'trasplantes', 'tag': 'NCMP000', 'prob': 0.0007549414348462665},
{'token': '.', 'tag': 'Fp', 'prob': 0.0959409594095941}]
```

Mostrar la oración etiquetada

El siguiente código te permite mostrar la oración etiquetada

```
In [107]: for palabra in oracion_etiquetada:
           print('{} / {}'.format(palabra['token'], palabra['tag']))
```

```
habla / VMIP3S0
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.

El etiquetado monosintáctico es correcto puesto que se genera en primera instancia un etiquetador monosintáctico basado en el corpus indicado, este corpus que es como un dataset según lo explicado en la clase entregara las características propias que se enmarcan desde el lenguaje, que para este caso es el español, las probabilidades dadas tanto para la transmisión como para emisión u observación son similares a las trabajadas y vistas en el material presentado, pero pueden cambiar, según los resultados las incidencias presentan porcentajes bajos, pero esto se debe al conjunto de multiplicaciones de las probabilidades que hacen que se venan levemente menores, pero si nos dirigimos a las etiquetas eagles se puede evidenciar que este etiquetado en un porcentaje cumple con lo relacionado en la tabla, como es el caso del verbo hablar, para el caso de trasplantes lo marca como nombre, y en otros correctos para el caso de grave que lo marca como adjetivo

- 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.

El enfermo grave habla de trasplantes

```
habla / VMIP3S0 con / SPS00 el / DA0MS0 enfermo / NCMS000 grave / AQ0CS0 de / SPS00
trasplantes / NCMP000
```

Como se habla en el apartado anterior es correcto el etiqueta hasta cierto punto, puesto que en la comparación de cada uno de los resultados las probabilidades propenden a ser bajas pero esto se debe a los resultados, y si se revisa lo presentado por las etiquetas eagles algunos se encuentran enmarcados en una solución óptima como grave y habla, otros como enfermo y trasplantes no se encuentran enmarcados en los que presenta la referencia de la etiqueta eagles, pero todo se debe a la forma como se presenta la oración, puesto que como se vio en la clase con el ejemplo

del vino los resultados pueden cambiar dependiendo la manera como el etiquetado trabaje sobre la oración, pero los resultados son favorables desde la construcción del etiquetador mediante el corpus hasta el etiquetado de la oración de estudio, con el punto y sin el punto.

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

Las limitaciones dependen directamente del análisis sobre el algoritmo, y la selección de cada uno de los métodos dispuestos para tal fin, se puede percibir las limitaciones del algoritmo si los resultados se cruzan con variaciones en las frases, donde se pueda realizar una validación cruzada como la vista en clase de los resultados en cada una de las fases. Otro detalle relevante del ejercicio es que al tratarse de un algoritmo genérico para la construcción del etiquetador y el etiquetado puede verse limitado en la modificación a fondo de las estructuras de código dispuesta para el cálculo de cada uno de los requerimientos.

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

Dentro de las posibles mejoras se puede resaltar el uso de técnicas vistas en otros cursos que apoyen el proceso de cálculo de las probabilidades, además de la adecuación mediante otras técnicas de la determinación de los mejores resultados para cada una de las fases, que hagan que el algoritmo sea mucho más corto y rápido. Otra mejora puede ser la síntesis del algoritmo, la creación de un etiquetador con la menor cantidad de sentencias posibles hacen que la capacidad de modificación sea mayor además de la prueba o cruce con diferentes corpus que mejoren los resultados observados en cada una de las fases. El uso de los autómatas es práctico para el ejercicio pero como se vio en el material de estudio el cruzarlo con herramientas de aprendizaje automático podría entregar resultados mejores en cuanto al proceso de etiquetado de una oración de estudio, con un apartado de etiquetas de Eagles como se trabajó en esta actividad.

In []:

In []: