Asignatura	Datos del alumno	Fecha
Aprendizaje Automático	Bernal Castillo Aldo Alberto Domínguez Espinoza Edgar Uriel Valdivia Medina Frank Edil	2 de agosto de 2022

Actividad 2: Análisis sintáctico

Librerías a utilizar

Para la implementación del algoritmo CKY solo serán necesarias tres paquetes adicionales, el primero (`itemgetter`) permite seleccionar un elemento de una lista a partir del índice, el segundo (`pprint`) mostrará de manera más clara los resultados de los árboles y finalmente, pandas que permitirá mostrar la matriz CKY en forma tabular, como un dataframe.

Datos de entrada

La variable `grammar_file` que aparece en el código contendrá las reglas gramaticales escritas en un archivo de texto. En este caso se han escrito las reglas propias del ejercicio, pero pueden variar (ver notebook adjunto).

La función `read_rules()` (ver notebook adjunto) recibe como argumento un texto con reglas gramaticales y probabilidades para convierlas en tres diccionarios que permiten buscar los datos necesarios para completar los algoritmos CKY y PCKY.

Se crea el lexicón, de forma común, en lingüística se dice que un elemento del lexicón en la entrada de un diccionario, entonces se define como tal en el código.

Se expresan las reglas gramaticales y léxicas en forma de tuplas como claves de una probabilidad que se usará para obtener las probabilidades de cada árbol obtenido por el algoritmo. Se eligen tuplas porque son fáciles de manejar, similar a los lenguajes Lisp.

Algoritmos CKY y PCKY

Ahora se implementa el algoritmo CKY, mismo que recibe como entrada una oración en forma de cadena de texto, un diccionario con reglas gramaticales y un lexicón adecuado a la gramática. Opcionalmente, se puede definir el argumento de entrada `verbose` al valor `True`, de esta manera se mostrará en pantalla cada uno de los pasos que sigue el algoritmo para crear la matriz correspondiente. Esta matriz también tendrá forma de diccionario (ver notebook anexo).

Enseguida se crean dos funciones, la primera, `get_prob()`, se usará para obtener las probabilidades del diccionario de probabilidades, la segunda, `update_prob()`, se usará para actualizar las probabilidades desde los nodos intermedios hasta el nodo padre (ver notebook anexo).

Asignatura	Datos del alumno	Fecha
Aprendizaje Automático	Bernal Castillo Aldo Alberto Domínguez Espinoza Edgar Uriel Valdivia Medina Frank Edil	2 de agosto de 2022

Se actualiza `cky_parser()`. Ahora `pcky_parser()` agregará la información de la probabilidad a cada lista agregada a `matrix_pcky`, con esto se completa el algoritmo solicitado por la práctica.

[ ]:
```
def pcky_parser(sentence, grammar, lexicon, probabilities_table,␣
↪verbose=False):
    words = sentence.lower().split()
    if verbose:
        print(words)
    matrix_pcky = dict()
    for i in range(len(words)):
        ordinate = 0
        if verbose:
            print("0: i=",i)
        for abscissa in range(i+1,len(words)+1):
            matrix_pcky[(ordinate, abscissa)] = list()
            if verbose:
                print("1: abscissa=",abscissa,"ordinate=",ordinate,"a-
o=",abscissa-ordinate)
            if(abscissa-ordinate == 1):
                for key in lexicon:
                    if verbose:
                        print("2: key=",key,"word=",words[ordinate])
                    if(words[ordinate] in lexicon[key]):
                        matrix_pcky[(ordinate,abscissa)].append(
                            (key,0,words[ordinate],words[ordinate],
                             ␣
↪get_prob((key,words[ordinate]),probabilities_table)))
                        if verbose:
                            print("2:",matrix_pcky)
            elif(abscissa-ordinate > 1):
                if verbose:
```

Asignatura	Datos del alumno	Fecha
Aprendizaje Automático	Bernal Castillo Aldo Alberto Domínguez Espinoza Edgar Uriel Valdivia Medina Frank Edil	2 de agosto de 2022

```
                    print("3:","(",ordinate,",",abscissa,")")
                for index in range(abscissa-ordinate-1):
                    left = matrix_pcky[(ordinate,abscissa-1-index)]
                    down = matrix_pcky[(abscissa-1-index,abscissa)]
                    if verbose:
                        print("4: index=",index)
                        print("4: left=",left,"\n  ","down=",down)
                    if not left or not down:
                        if verbose:
                            print("5: Nothing to do")
                    else:
                        for a in left:
                            for b in down:
                                if((a[0],b[0]) in grammar):
                                    matrix_pcky[(ordinate,abscissa)].
↪append(
                                        (grammar[(a[0],b[0])][0],
                                        abscissa-1-index,a[0],b[0],
                                    ␣
↪update_prob(a,b,grammar[(a[0],b[0])][0],probabilities_table))
                                        )
                                    if verbose:
                                        print("6:␣
↪add=",grammar[(a[0],b[0])][0],abscissa-1-index,a[0],b[0])
                                else:
                                    if verbose:
                                        print("6: Nothing to do")
            if verbose:
                print(matrix_pcky)
                print("9: Next Ordinate")
            ordinate+=1
```

Asignatura	Datos del alumno	Fecha
Aprendizaje Automático	Bernal Castillo Aldo Alberto Domínguez Espinoza Edgar Uriel Valdivia Medina Frank Edil	2 de agosto de 2022

```
    return matrix_pcky
```

Obtención de resultados

La función `tree()` recorre una matriz generada por el algoritmo CKY en busca de árboles a partir del valor de un nodo en particular, tomando en cuenta una ruta a través de un índice (llamado backpointer en el algoritmo) creado por el algoritmo.

[ ]:
```
def tree (matrix, cell, parent="S", index=0, pcky=False):
    list = []
    if len(matrix[cell]) == 0:
        return list
    if matrix[cell][0][1] == 0:
        if matrix[cell][index][0] == parent:
            if pcky:
                return␣
↪[(matrix[cell][index][0],matrix[cell][index][4]),
                        matrix[cell][index][2]]
            else:
                return [matrix[cell][index][0], matrix[cell][index][2]]
        else:
            if index+1 < len(matrix[cell]):
                if pcky:
                    return tree(matrix,cell,parent,index+1,pcky=True)
                else:
                    return tree(matrix,cell,parent,index+1)
            else:
                return list
    elif (matrix[cell][index][0]) == parent:
        if pcky:
            list.append((matrix[cell][index][0],matrix[cell][index][4]))
        else:
            list.append(matrix[cell][index][0])
```

```
        child = []
        if pcky:
            child.append(
                tree(matrix,
                     (cell[0],matrix[cell][index][1]),
                     matrix[cell][index][2],pcky=True))
        else:
            child.append(
                tree(matrix,
                     (cell[0],matrix[cell][index][1]),
                     matrix[cell][index][2]))
        if pcky:
            child.append(
                tree(matrix,
                     (matrix[cell][index][1],cell[1]),
                     matrix[cell][index][3],pcky=True))
        else:
            child.append(
                tree(matrix,
                     (matrix[cell][index][1],cell[1]),
                     matrix[cell][index][3]))
        list.append(child)
        return list
    else:
        if index+1 < len(matrix[cell]):
            if pcky:
                return tree(matrix,cell,parent,index+1,pcky=True)
            else:
                return tree(matrix,cell,parent,index+1)
        else:
            return list
```

Asignatura	Datos del alumno	Fecha
Aprendizaje Automático	Bernal Castillo Aldo Alberto Domínguez Espinoza Edgar Uriel Valdivia Medina Frank Edil	2 de agosto de 2022

La función `find_solutions()` servirá como *frontend* para todas las funciones anteriores, brindará la respuesta propuesta por el algoritmo y la matriz correspondiente. Dependerá de los argumentos si usará CKY o PCKY (ver notebook adjunto).

[ ]:
```
grammar_rules, lexicon_rules, probabilities = read_rules(grammar_file)
```

[ ]:
```
options, matrix = find_solutions("Time flies like an arrow",
                         grammar_rules,lexicon_rules,"S")
pprint(options)
```

```
[['S',
  [['NP', [['Nominal', 'time'], ['Nominal', 'flies']]],
   ['VP', [['Verb', 'like'], ['NP', [['Det', 'an'], ['Nominal',␣
↪'arrow']]]]]]],
 ['S',
  [['NP', 'time'],
   ['VP',
    [['Verb', 'flies'],
     ['PP',
      [['Preposition', 'like'],
       ['NP', [['Det', 'an'], ['Nominal', 'arrow']]]]]]]]]]
```

[ ]:
```
options, matrix = find_solutions("Time flies like an arrow",
                         grammar_rules,lexicon_rules,"S",probabilities)
pprint(options)
```

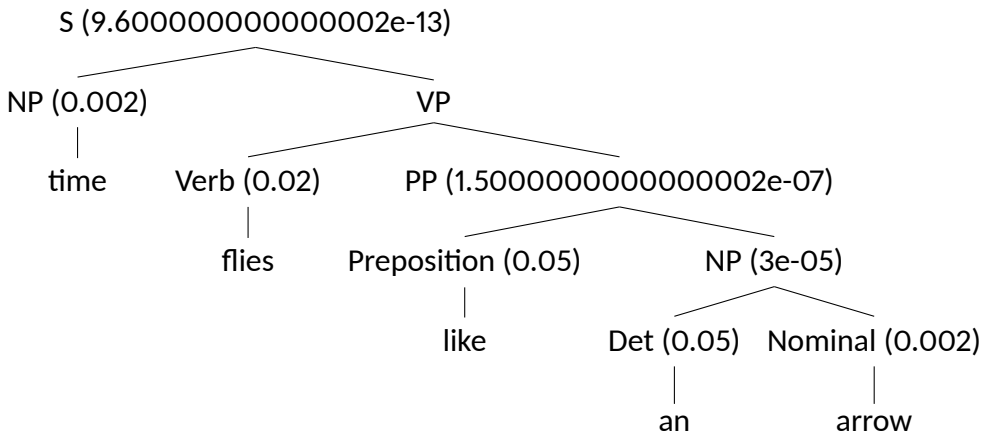
```
[('S', 9.600000000000002e-13),
 [[('NP', 0.002), 'time'],
  [('VP', 6.000000000000001e-10),
   [[('Verb', 0.02), 'flies'],
    [('PP', 1.5000000000000002e-07),
     [[('Preposition', 0.05), 'like'],
      [('NP', 3e-05),
       [[('Det', 0.05), 'an'], [('Nominal', 0.002), 'arrow']]]]]]]]]
```

Asignatura	Datos del alumno	Fecha
Aprendizaje Automático	Bernal Castillo Aldo Alberto Domínguez Espinoza Edgar Uriel Valdivia Medina Frank Edil	2 de agosto de 2022

Cuestionario

1. ¿Es correcto el análisis sintáctico que se ha obtenido? Justifica la respuesta.

Sí. Basado en la gramática proporcionada el resultado es satisfactorio y se muestra a continuación.



2. ¿Cuáles son las limitaciones de aplicar el algoritmo CKY probabilístico para realizar el análisis sintáctico? Justifica la respuesta.

Los problemas se presentan debido a la información guardada en la gramática, si ésta no tiene un elemento particular necesario para el análisis, proporcionará resultados truncos.

Por supuesto, las probabilidades de aparición de la gramática podrían llevar a resultados dudosos, incluso cuando su objetivo es el contrario, pero estas probabilidades siempre dependerán del corpus de origen, que puede ser distinto al de la oración a analizar, ya sea en registro, en dialecto, etc.

En el caso particular de este ejercicio, el hecho de tener algunos símbolos intermedios creados para llegar a la forma normal de Chomsky permite que el algoritmo genere posibles soluciones impensables para el análisis linguístico. El algoritmo CKY propone dos soluciones y el PCKY seleccionó el correcto, pero si se examina la opción desechada se puede observar:

```
['NP', [['Nominal', 'time'], ['Nominal', 'flies']]]
```

Lo cual es un elemento impensable pues sugiere que una frase nomimal puede tener dos núcleos. Este análisis va en contra del principio de endocentrismo que establece, como su nombre indica, que cada frase tiene un núcleo que determina el tipo de la frase.

Finalmente, el algoritmo que usa este tipo de gramáticas, arrastra la misma gran crítica que los lingüistas hacen a la teoría de Chomsky: No es capaz de analizar lenguas aglutinantes.

3. ¿Qué posibles mejoras que se podrían aplicar para mejorar el rendimiento del análisis sintáctico? Justifica la respuesta.

El algoritmo se basa en el uso de cualquier gramática en forma normal de Chomsky, esto es porque computacionalmente es posible generar automáticamente gramáticas mediante corpus de análisis, sin embargo, el programa minimalista de Chomsky sugiere la teoría X-barra que tiene su principio en los estudios de Ray Jackendoff. Esta teoría proporciona la posibilidad de crear gramáticas consistentes siempre en la forma normal de Chomsky. Dejar de ignorar el conocimiento linguístico proporcionaría mejores gramáticas para la IA.

Por otra parte, analizar solo las probabilidades de los subárboles que llevan a una propuesta de solución aumentaría el rendimiento a nivel de cómputo.