UNIVERSITAT POLITÈCNICA DE VALÈNCIA

MIARFID
Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

DSIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital
Universitat Politècnica de València

Boletín de ejercicios (I)

Predicción Estructurada Estadística

Autor: Juan Antonio López Ramírez

Curso 2019-2020

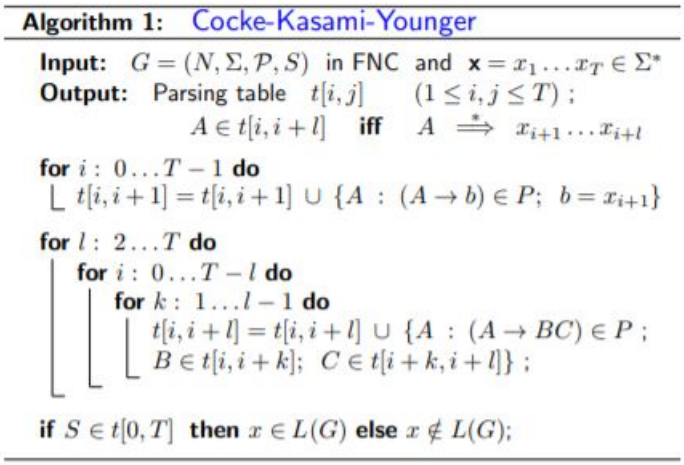

Figura 1: Algoritmo *Cocke-Kasami-Younger*.

Teoría

Ejercicio 5

En este ejercicio, se pide diseñar un algoritmo que, dada una gramática en forma normal de Chomsky y una secuencia de entrada x, obtenga el número de árboles de análisis para x.

El algoritmo de *Cocke-Kasami-Younger* visto en clase de teoría obtiene, para una gramática en forma normal de Chomsky y una palabra de entrada x, si x pertenece al lenguaje que genera la gramática. El diseño de este algoritmo se puede apreciar en la figura 1.

En la figura 2 se puede observar la tabla resultado de aplicar CKY sobre una determinada gramática y una palabra. Como se puede apreciar en la celda superior derecha, hay dos elementos que empiezan por el axioma de la gramática, lo que indica que hay dos posibles árboles de análisis para *aab* y la gramática.

El algoritmo que pide el ejercicio 5 sería una pequeña modificación de CKY, pero en lugar de la última línea de código en la que se comprueba si el axioma está en la celda superior derecha; habría que recorrer todos los elementos de dicha celda y que vaya contando aquellos que contienen el axioma.

Una forma de hacer esto en código python (suponiendo que cada elemento de la celda es una tupla donde el primer elemento es el no terminal y S es el axioma de la gramática) sería:

```
cont = 0
for x in t[0, T]:
    if S==x[0]:
        cont += 1
return cont
```

De esta forma, si la palabra no pertenece al lenguaje generado por la gramática, el resultado del algoritmo modificado es 0, ya que no hay ningún árbol de análisis para esa palabra con esa gramática.

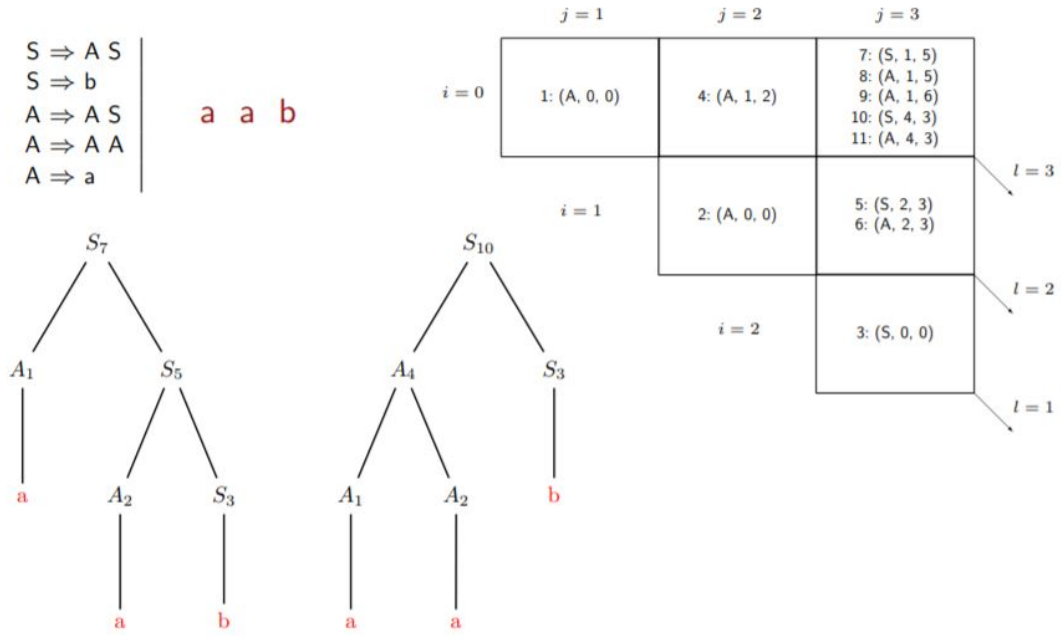

Figura 2: Resultados de CKY para la palabra *aab* y la gramática *G*.

Ejercicio 6

En este ejercicio, se pide modificar el algoritmo de Viterbi basado en CKY que se ha visto en la clase de teoría. Este algoritmo calcula la probabilidad del árbol más probable para una secuencia de entrada x, de longitud T.

La inicialización de Viterbi, para todo A perteneciente a los símbolos no terminales y para todo i entre 0 y T-1, es de la siguiente forma:

$$e(A < i, i+1 >) = p(A \to b)\delta(b, x_{i+1}) \tag{1}$$

Y el paso de recursión, teniendo en cuenta que l va desde 2 hasta T para todo i desde 0 hasta T - l, es:

$$e(A < i, i+l >) = \max_{B,C \in N} p(A \to BC) \max_{k=1,\ldots,l-1} e(B < i, i+k >) e(C < i+k, i+l >) \tag{2}$$

Por tanto, la probabilidad del árbol de análisis con máxima probabilidad para la cadena x es:

$$e(S < 0, T >) \tag{3}$$

Si tenemos en cuenta que las gramáticas que dice el ejercicio generan siempre un nodo terminal por el nodo no terminal de la izquierda (B en este caso), entonces habría que modificar el paso de recursión para que k sea siempre 1, ya que la primera llamada recursiva siempre recubre un nodo terminal. Por tanto, quedaría como sigue:

$$e(A < i, i+l >) = \max_{B,C \in N} p(A \to BC) e(B < i, i+1 >) e(C < i+1, i+l >) \tag{4}$$

El coste temporal del algoritmo original es N^2T, siendo T la longitud de la cadena de entrada y N el número de símbolos no terminales de la gramática. Como ahora se ha suprimido el segundo máximo, el coste del algoritmo de la solución es N^2. Es decir, el antiguo tenía coste cúbico, mientras que el nuevo tiene coste cuadrático.

Prácticas

En esta sección, se va a explicar el proceso que se ha seguido para resolver el problema de prácticas que se presenta en el boletín de ejercicios.

Para empezar, hay 3 corpus (TS-EQ, TS-IS y TS-SC), cada uno con palabras que representan los triángulos de una determinada clase: equiláteros, isósceles y escalenos.

Además, hay 3 modelos G1, G2 y G3, cada uno con 3 grámaticas, una por cada tipo de triángulo. Es decir, se tienen nueve submodelos (G1-EQ, G1-IS, G1-SC, G2-EQ...).

Con la herramienta scfg-toolkit se puede calcular la probabilidad de que una palabra x sea de uno de los tres tipos de triángulo. En este caso, si se le pasa un corpus TS con n palabras y un submodelo G, la herramienta dirá, para cada palabra del corpus, la probabilidad de pertenencia a la clase que especifica el submodelo (en total, devolverá n probabilidades).

Por último, se dispone de un script llamado *confus*, el cuál calcula la matriz de confusión entre clases en función del modelo utilizado, para así poder analizar y estudiar los resultados obtenidos en la clasificación.

Con todos estos elementos, se ha realizado un script llamado *scfg-practica.py* en el que se resuelven los dos apartados del ejercicio, es decir, la evaluación estadística y la clasificación.

Evaluación estadística

Para la evaluación estadística se ha usado la métrica del cálculo de la perplejidad, cuya fórmula es:

$$2^{\frac{-1}{N}\sum_x log_2 P_M(x)} \tag{5}$$

Donde $P_M(x)$ es la probabilidad asignada a la palabra x por el modelo M.

Esta probabilidad es la que calcula el toolkit, de modo que la parte del script que resuelve esto es:

```
for c in ["TS-EQ", "TS-IS", "TS-SC"]:
    for a in ["EQ","IS","SC"]:
        for g in ["G1","G2","G3"]:
                call("./scfg-toolkit/scfg_prob -g models/"+g+"-"+a+" -m corpus/
                    "+c+" > probabilities/res_"+c+"_"+g+"_"+a, shell=True)
```

Cada corpus c se evalúa con cada uno de los 9 submodelos. De esa forma, se obtienen 27 ficheros de probabilidades, que se guardan en la carpeta *probabilities*.

Una vez hecho esto, se calculan las 27 perplejidades asociadas a cada uno de los ficheros de *probabilities*. Esto se hace de la siguiente forma:

```
1  perplexities = []
2  for c in ["TS-EQ", "TS-IS", "TS-SC"]:
3      for g in ["G1","G2","G3"]:
4          for a in ["EQ","IS","SC"]:
5              f = open("./probabilities/res_"+c+"_"+g+"_"+a, "r")
6              fichero = f.read()[:-1]
7              sumatorio = 0
8              N = 0
9              for numero in fichero.split("\n"):
10                 num = float(numero)
11                 if num!=0:
12                     sumatorio += math.log2(num)
13                 N += 1
14             perplexity = math.pow(2, (-1/N)*sumatorio)
15             perplexities.append("Perplejidad en "+c+"_"+g+"_"+a+" es "+str(
                   perplexity))
16 with open("perplexities.txt", "w") as f:
17         for i in perplexities:
18             f.write(i+"\n")
```

Es decir, se crea una lista *perplexities* donde se guardarán las 27 perplejidades. Se recorre cada fichero, se obtiene el sumatorio de la ecuación de la perplejidad con el bucle de la línea 9 (obviando aquellas probabilidades iguales a 0, ya que no se puede calcular logaritmo de 0) y , una vez hecho esto, se calcula la perplejidad y se añade a la lista, indicando el conjunto de prueba y el submodelo utilizado. Finalmente, se crea un fichero donde se guardan las perplejidades de la lista, cada una en una línea de texto.

Los resultados obtenidos se pueden observar en las tablas 1, 2 y 3.

Submodelo	Conjunto de prueba	Perplejidad
G1-EQ	TS-EQ	99095.2
G1-EQ	TS-IS	191.8
G1-EQ	TS-SC	245.9
G1-IS	TS-EQ	47334.8
G1-IS	TS-IS	26581.8
G1-IS	TS-SC	21621.9
G1-SC	TS-EQ	48369.4
G1-SC	TS-IS	27508.9
G1-SC	TS-SC	33618.3

Tabla 1: Resultados obtenidos con el modelo G1.

Teniendo en cuenta que, a menor perplejidad, mayor tasa de acierto para los modelos, en las tablas se puede apreciar que todos ellos proporcionan un cierto error si hubiese que clasificar.

Por ejemplo, en el submodelo G1-EQ, la perplejidad más baja se obtiene para el corpus TS-IS (191.8), cuando debería hacerlo, a priori, con el TS-EQ (99095.2). Lo mismo sucede para los otros dos submodelos de G1.

Sin embargo, para G3, el único submodelo que parece que clasificaría erróneamente sería el G3-IS, que obtiene menor perplejidad con TS-SC, aunque la diferencia con TS-IS es relativamente corta, sobre todo en comparación con los resultados obtenidos en G1 y G2.

Submodelo	Conjunto de prueba	Perplejidad
G2-EQ	TS-EQ	635136.4
G2-EQ	TS-IS	389521.3
G2-EQ	TS-SC	520221.7
G2-IS	TS-EQ	102498.6
G2-IS	TS-IS	52217.6
G2-IS	TS-SC	49786
G2-SC	TS-EQ	68152.5
G2-SC	TS-IS	42723.1
G2-SC	TS-SC	43543.3

Tabla 2: Resultados obtenidos con el modelo G2.

Submodelo	Conjunto de prueba	Perplejidad
G3-EQ	TS-EQ	986.2
G3-EQ	TS-IS	550208.1
G3-EQ	TS-SC	1081980.3
G3-IS	TS-EQ	1283.3
G3-IS	TS-IS	1254.9
G3-IS	TS-SC	1218.7
G3-SC	TS-EQ	1272.7
G3-SC	TS-IS	1258.3
G3-SC	TS-SC	1218

Tabla 3: Resultados obtenidos con el modelo G3.

Clasificación

Para la tarea de clasificación, se parte de los ficheros de probabilidades obtenidos en la evaluación estadística. El objetivo es comprobar, para cada palabra de los corpus, en qué tipo de triángulo clasifica cada uno de los modelos, y compararlos para saber el que proporciona un error menor. Por ejemplo, para un modelo G y el corpus TS-EQ, el objetivo ideal sería que toda palabra de este, que obtiene 3 probabilidades distintas (una con G-EQ, otra con G-IS y otra con G-SC), obtuviera la mayor probabilidad con G-EQ.

Siguiendo esta filosofía, el código empleado para comprobar a qué tipo de triángulo se ha clasificado cada palabra es el siguiente:

```
for g in ["G1","G2","G3"]:
        for c in ["EQ", "IS", "SC"]:
            f1 = []
            f2 = []
            f3 = []
            i = 1
            for a in ["EQ","IS","SC"]:
                f = open("./probabilities/res_TS-"+c+"_"+g+"_"+a, "r")
                fichero = f.read()[:-1]
                for numero in fichero.split("\n"):
                    if i==1:
                        f1.append(float(numero))
                    elif i==2:
                        f2.append(float(numero))
                    else:
                        f3.append(float(numero))
                i+=1
            lista = zip(f1, f2, f3)
            res = []
            for tupla in lista:
                if max(tupla)==tupla[0]:
                    res.append("EQ")
                elif max(tupla)==tupla[1]:
                    res.append("IS")
                else:
                    res.append("SC")
            with open("fichero_res"+c+"_"+g+".txt", "w") as f:
                for i in res:
                    f.write(c+ " "+ i+"\n")
for g in ["G1","G2","G3"]:
        res = []
        for c in ["EQ", "IS", "SC"]:
            f = open("fichero_res"+c+"_"+g+".txt", "r")
            fichero = f.read()[:-1]
            for linea in fichero.split("\n"):
                res.append(linea)

        with open("fichero_res_def"+g+".txt", "w") as f:
            for i in res:
                f.write(i+"\n")

for g in ["G1","G2","G3"]:
        print("Resultados obtenidos con "+ g)
        call("./confus fichero_res_def"+g+".txt" ,shell=True)
        print("\n")
```

Primeramente, se recorren los 9 submodelos y se crean 3 listas (f1, f2 y f3) donde se guardarán las probabilidades asociadas a cada tipo de triángulo (en f1 se guardan las probabilidades obtenidas con TS-EQ, en f2 las de TS-IS y en f3 las de TS-SC).

```
jalopez@DESKTOP-HIPIB76:/mnt/d/Cosas_de_Juan/PEE/PCFG-triangles$ python3 scfg-practica.py
Resultados obtenidos con G1
        EQ   IS   SC  Err Err%
   EQ  597  285  118  403 40.3
   IS   88  471  441  529 52.9
   SC   71  406  523  477 47.7

Error: 1409 / 3000 = 46.97 %


Resultados obtenidos con G2
        EQ   IS   SC  Err Err%
   EQ  281  211  508  719 71.9
   IS   71  215  714  785 78.5
   SC   81  190  729  271 27.1

Error: 1775 / 3000 = 59.17 %


Resultados obtenidos con G3
        EQ   IS   SC  Err Err%
   EQ  789  102  109  211 21.1
   IS  178  512  310  488 48.8
   SC  106  421  473  527 52.7

Error: 1226 / 3000 = 40.87 %
```

Figura 3: Matrices de confusión y errores de clasificación obtenidos con *scfg-practica.py*.

Una vez hecho esto, se hace un *zip* de las tres listas, obteniendo una lista de tuplas de tres elementos. Esto quiere decir que cada elemento de la lista se corresponde con una palabra y la tupla contiene sus tres probabilidades.

Como cada probabilidad de la tupla está ordenada en función del tipo de triángulo (EQ, IS y SC), se recorre la lista y, si el máximo es el primer elemento de la tupla, quiere decir que el modelo lo ha clasificado en EQ; si es el segundo, en IS; y si es el tercero, en SC. Por tanto, se añade el tipo de triangulo a la lista *res* y se guardan en un fichero llamado *fichero res* las palabras a clasificar junto con las clasificaciones obtenidas para cada palabra. De esta forma, tendríamos 9 ficheros resultado, tres de cada corpus con su clasificación por los tres modelos que se han utilizado.

Como solo nos interesan tres ficheros, uno por cada modelo, donde dentro se indique el tipo de triángulo y la clasificación que se ha obtenido, se realiza el bucle de la línea 30, guardando los resultados en un fichero llamado *fichero res def*.

Finalmente, se llama al confus con cada *fichero res def* para obtener las tres matrices de confusión y los tres errores de clasificación, uno por cada modelo G1, G2 y G3.

Los resultados obtenidos se pueden apreciar en la figura 3. Como se puede observar, el menor error se consigue con el modelo G3, lo que concuerda con los resultados de la evaluación estadística.