Instituto Tecnológico y Estudios Superiores de Monterrey
Víctor Hugo Franco Juárez A01366475
Juan Pablo Ortiz Ortega A01366969

Actividad 4.5 Reflexión de Similitud en Textos

Situación Problema

La detección de similitud en textos, específicamente enfocado en un contexto académico en la detección de plagio, es un problema que ha llevado a los computólogos a desarrollar sistemas para la detección de similitudes en texto utilizando diferentes técnicas. El proyecto desarrollado consistió en utilizar una de estas técnicas e implementarlas en un programa de Python para detectar qué tan similares son dos textos, en este caso programas en el lenguaje *C-* (C Minus), y poder determinar si es que ambos programas fueron realizados por un mismo autor/se está realizando plagio o robo de código.

La técnica que utilizamos para el desarrollo de este proyecto fue la vectorización, la cual consiste en colocar las palabras del texto a evaluar. En el caso de un programa de computadora, existen ciertas palabras y símbolos que determinan su funcionamiento, lo cual provoca que el "vocabulario" del lenguaje, esté limitado por el número de palabras y símbolos (tokens) del lenguaje de programación. En la implementación realizada, la vectorización se realizó sobre los 32 tokens que conforman el léxico de C-, los cuales son detectados por un analizador léxico desarrollado anteriormente por el equipo del proyecto. Junto con la vectorización, se utilizaron los métodos *tf* (Term Frequency) y el *tf-idf* (Inverse Document Frequency). para detectar la frecuencia de aparición de los tokens en los programas a comparar, y los resultados fueron analizados utilizando comparación de cosenos, obteniendo un valor entre 0 y 1 que representa la similitud entre ambos programas.

Preparación de programas/textos

```
def prepareTexts():
    global bagA, bagB, firstCode, secondCode, mixedBag, dictA,
dictB
    #Separa todos los tokens de los programas
    bagA = firstCode.split(' ')
    bagB = secondCode.split(' ')
    tokenBag = {Todos los tokens del lenguaje C-}
    #Llena los diccionarios con el no. de veces que aparece el
    token en cada programa
    dictA = dict.fromkeys(mixedBag, 0)
    for token in bagA:
        dictA[token]+=1
```

```
    dictB = dict.fromkeys(mixedBag, 0)
    for token in bagB:
        dictB[token]+=1
```

El código separa los tokens de ambos programas en dos “bolsas” de palabras, y a la vez que genera un vector *tokenBag* el cual contiene todos los 32 tokens existentes en el lenguaje C-. Posteriormente se crean dos diccionarios a partir de las bolsas de palabras de los códigos, donde la llave es el token y su valor es el número de veces que este aparece en el código. De esta forma, podemos asignar una medida a cada una de las palabras.

Implementación del tf

```
#Función TF calcula el TF de los programas. Recibe el diccionario
generado en prepareTexts y la bolsa de palabras
# del mismo programa. Regresa un diccionario con el TF.
def TF(dictionary, bag):
    tfDict = {} #Genera diccionario con el TF
    bagCount = len(bag)
    for token, count in dictionary.items():
        tfDict[token] = count / float(bagCount)
    return tfDict
```

La implementación del TF cuenta el número de veces que aparece la palabra en el código original y no utiliza la bolsa de tokens totales.

Implementación del tf-idf

```
#Función IDF calcula el IDF de los programas. Recibe los programas
a utilizar.
# Regresa un diccionario con el IDF.
def IDF(programs):
    N = len(programs)
    idfDict = {} #Genera el diccionario del IDF
    idfDict = dict.fromkeys(programs[0].keys(), 0)
    for program in programs:
        for token, value in program.items():
            if value > 0:
                idfDict[token] += 1
    for token, value in idfDict.items():
            idfDict[token] = (math.log(((1+N) /
(1+float(value))))+1 #+1 obtenido de scikit learn

    return idfDict
```

Para la implementación del IDF, se utilizó la adecuación a la fórmula propuesta por Scikit-learn, con el objetivo de que nunca se realice una división entre 0.

Finalmente, se realiza el cálculo del TF-IDF con la siguiente implementación, que representa la ecuación $tf - idf(t, D) = tf(t, D) * idf(t, D)$

```
#Función TFIDF genera el TFIDF. recibe el TF del programa y los
IDFs.
#Regresa el diccionario del TFIDF.
def TFIDF(tf, idfs):
    tfidf = {} #Genera el diccionario del TFIDF
    for token, value in tf.items():
        tfidf[token] = value * idfs[token]

    return tfidf
```

Finalmente, para comparar la similitud entre los dos textos a partir de los vectores generados se calcula el coseno del ángulo que estos forman:

```
#Imprime el resultado
def calculate(method):
    #Abre los códigos, prepara los textos
    openCode(1)
    openCode(2)
    prepareTexts()

    #Calcula los TF
    firstTF = TF(dictA, bagA)
    secondTF = TF(dictB, bagB)

    #Si method = 1, calcula el TF-IDF e imprime la comparación de
cosenos usando TF-IDF
    if method == 1:
        idfs = IDF([dictA, dictB])

        firstTFIDF = TFIDF(firstTF, idfs)
        secondTFIDF = TFIDF(secondTF, idfs)

        finalArray1 = dictToArray(firstTFIDF)
        finalArray2 = dictToArray(secondTFIDF)


        cosine = cosine_similarity(finalArray1,finalArray2)

        print("Similaridad de coseno usando TF-IDF:",cosine)

    #Si method = 2, imprime la comparación de cosenos usando TF
```

```
    if method == 2:
        finalArray1 = dictToArray(firstTF)
        finalArray2 = dictToArray(secondTF)

        cosine = cosine_similarity(finalArray1,finalArray2)

        print("Similaridad de coseno usando TF:",cosine)
```

La función *calculate()* genera el cálculo del coseno del ángulo generado por ambos métodos. Esta hace uso de la función *cosine_similarity()*, la cual pertenece a la librería de *sklearn*. El coseno que es calculado a partir de este método nos da la similitud entre ambos programas, el cual utilizamos como métrica para conocer qué tan similares son ambos y, consecuentemente, determinar si es que ambos programas pertenecen al mismo autor y se trata de un caso de plagio.

Se realizaron cuatro casos de prueba para evaluar la implementación:

Casos de prueba

Caso de prueba 1: Códigos completamente iguales
Para el caso de prueba #1, se realizó la comparación entre dos archivos .c- que contienen el siguiente código (ambos):

```
/* Un programa para realizar ordenación por
selección en un arreglo de 10 elementos. */
int x[10];

int minloc ( int a[], int low, int high )
{ int i; int x; int k;
  k = low;
  x = a[low];
  i = low + 1;
  while (i < high)
  { if (a[i] < x)
      { x = a[i];
        k = i;  }
    i = i + 1;
  }
  return k;
}

void sort( int a[], int low, int high)
```

Se obtuvo el siguiente resultado:

```
Similaridad de coseno usando TF-IDF: [[1.]]
Similaridad de coseno usando TF: [[1.]]
```

Predecible y correctamente, el resultado es 1.0 (100% de similitud) según ambos métodos.

Caso de prueba 2: Mismo programa, diferente nombre de ID y diferente valor numérico

En el caso de prueba #2, se probaron dos códigos que cuentan con los mismos Tokens (incluso sus posiciones son las mismas) pero cuentan con diferentes nombres (IDs) y diferentes valores numéricos.

Código 1:

```
/* Un programa para realizar ordenación por
selección en un arreglo de 10 elementos. */
int x[10];

int minloc ( int a[], int low, int high )
{ int i; int x; int k;
  k = low;
  x = a[low];
  i = low + 1;
  while (i < high)
  { if (a[i] < x)
      { x = a[i];
        k = i;  }
    i = i + 1;
  }
  return k;
}

void sort( int a[], int low, int high)
```

Código 2:

```
/* Un programa para realizar ordenación por
selección en un arreglo de 10 elementos. */
int x[10];

int hola ( int e[], int bajo, int alto )
{ int q; int r; int l;
  l = low;
  r = a[low];
  q = low + 8;
  while (q < high)
  { if (e[q] < x)
      { r = e[q];
        l = r;  }
    l = l + 8;
  }
  return q;
}

void sort( int t[], int alto, int bajo)
```

Se obtuvo el siguiente resultado:

```
Similaridad de coseno usando TF-IDF: [[1.]]
Similaridad de coseno usando TF: [[1.]]
```

Nuevamente, el resultado fue de 1.0 (100%), ya que estos métodos nos limitan a la detección de los 32 tokens que comprenden el léxico del lenguaje, por lo que las variaciones en nombres de ID y valores numéricos no son tomados en cuenta ya que sólo nos interesa el tipo de dato que son. En un caso realista, esto puede indicar un posible plagio o robo de trabajo académico.

Caso de prueba 3: Misma función, diferentes usos

En el caso de prueba 3, se compararon dos programas que cuentan con una función idéntica descrita al inicio, pero que se utiliza en distintos contextos.

Código 1:

```
void imprimeError()
{
    print(Error)
}

int main()
{
    a = 35;
    b = 57;

    if (b-a < 0){
        imprimeError();
    }

}
```

Código 2:

```
void imprimeError()
{
    print(Error)
}


int minloc ( int a[], int low, int high )
{ int i; int x; int k;
  k = low;
  x = a[low];
  i = low + 1;
  while (i < high)
  { if (a[i] < x)
      { x = a[i];
        k = i;  }
    i = i + 1;
  }
  if (k == x)
  {
    imprimeError()
  }
  else{
    return k;
  }
  return k;
```

```
}

void sort( int a[], int low, int high)
```

Se obtuvo el siguiente resultado:

```
Similaridad de coseno usando TF-IDF: [[0.89418894]]
Similaridad de coseno usando TF: [[0.91400314]]
```

Es aquí donde comenzamos a ver las limitaciones del uso de estos métodos para determinar la similitud en programas, ya que, a simple vista, podemos observar que ambos programas son distintos, y la función reutilizada representa una pequeña parte del código, no obstante, el resultado sugiere que los programas son altamente similares en un 89 y 91 por ciento correspondientemente, lo cual sugiere que es altamente posible que este sea un caso de plagio.

Caso de prueba 4: Códigos completamente distintos

En este caso, se compararon dos códigos completamente distintos y de longitudes distintas.

Código 1:

```
void imprimeError()
{
    print(Error)
}

int main()
{
    a = 35;
    b = 57;

    if (b-a < 0){
        imprimeError();
    }
}

void imprimeError()
{
    print(Error)
}


int minloc ( int a[], int low, int high )
{ int i; int x; int k;
  k = low;
  x = a[low];
  i = low + 1;
  while (i < high)
  { if (a[i] < x)
      { x = a[i];
        k = i;  }
```

```
    i = i + 1;
  }
  if (k == x)
  {
    imprimeError()
  }
  else{
    return k;
  }
  return k;

}

void sort( int a[], int low, int high)
```

Código 2:

```
int mult(n)
{
   return n*n
}
```

Se obtuvo el siguiente resultado:

```
Similaridad de coseno usando TF-IDF: [[0.75387132]]
Similaridad de coseno usando TF: [[0.84594224]]
```

En este caso, podemos observar a simple vista que ambos programas son completamente diferentes. Si se diera la tarea de determinar qué tan similares son ambos programas a una persona, la respuesta debería ser muy cercana al 0%, no obstante, los resultados generados por estos métodos sugieren una similaridad por lo menos mayor al 75%.

Conclusiones

El resultado de la implementación utilizando TF, TF-IDF y similitud de cosenos **no** es una métrica ideal para determinar plagio en programas. Eso se debe a que son métodos para comparar textos en lenguaje humano y con vocabularios amplios, no códigos de un lenguaje de programación con un léxico limitado (C- cuenta con 32 tokens únicamente), ya que a pesar de que pueda haber una gran diferencia en longitud y, por consiguiente, una gran diferencia en la frecuencia con la que estas palabras son utilizadas, al final se estará utilizando los mismos 32 tokens del léxico limitado del lenguaje de programación, por lo que los resultados obtenidos por ambos métodos siempre van a ser altos en una escala entre 0 y 1 (obtenida por la comparación de cosenos).

En clase se proponía originalmente que el valor obtenido de la escala equivalía al porcentaje de similitud (0.5 = 50%, 0.98 = 98%) sin embargo, los casos de prueba muestran valores altos/tendiendo a uno sin importar la similaridad de los códigos.

Para poder utilizar la implementación para evaluar similaridad y posibles plagios, la escala se tendría que adaptar a considerar valores cercanos a 0.7 como NADA PARECIDOS, y valores de más de 0.95 como parecidos. No obstante, esta nueva escala puede ser confusa para usuarios acostumbrados a interpretar el 0-1 como porcentajes.

Detectar similaridad entre los códigos de los programas es una tarea muy compleja, la cual va más allá de los métodos existentes para comparar textos convencionales, en parte por las razones mencionadas anteriormente y en parte por las numerosas formas en que los programadores pueden evadir los métodos de similitud en código, los cuales surgen como consecuencia de las limitaciones de los métodos utilizados en textos convencionales. En realidad, la forma más eficiente de detectar correctamente plagio en código es por medio de revisiones hechas por programadores expertos, sin embargo, es un método manual y que consume tiempo, por lo que se sigue en búsqueda de alternativas computacionales para automatizar este proceso.

Referencias:

Anirudha Simha, Principle Associate Software Engineer, Kai Chatbot Team. (2021). Understanding TF-IDF for Machine Learning. Capital One. https://www.capitalone.com/tech/machine-learning/understanding-tf-idf/