PageRank
y TextRank

Samuel Cifuentes García - UO238524

Sistemas de Información para la Web

1 PageRank

Dadas una lista de aristas en forma de [[A, B], [A, C], ...] que representan un grafo sobre el que se pide aplicar PageRank. Para representar este grafo creo la clase Graph en el fichero common.py.

Listing 1.1. Creación del grafo

```
class Graph:
    def __init__(self, edges, undirected=False):
        # Keep a set with every node to refer to them via index
        nodes = set()
        for edge in edges:
            nodes.add(edge[0])
            nodes.add(edge[1])
        nodes = sorted(list(nodes))
        n = len(nodes)
        self.nodes = nodes

        # Generates the adyacency matrix from the edge list
        matrix = np.zeros((n, n))
        for a, b in edges:
            i, j = nodes.index(a), nodes.index(b)
            matrix[j][i] = 1

        if undirected: # For undirected graphs
            # This calculates the symmetrical matrix
            matrix = matrix + matrix.T - np.diag(matrix.diagonal())

        self.m = self.__normalize_matrix(matrix)
```

Creo un set de nodos para guardar el nombre de los nodos, A, B, C... y poder referirme a ellos a través de los índices a la hora de trabajar con matrices. Utilizaré numpy para trabajar con matrices. En el siguiente paso genero la matriz de adyacencia a partir de las aristas dadas Ignoremos lo de undirected por ahora, en TextRank explicaré su función Por último normalizo la matriz. Es decir, si partimos de la matriz de adyacencia:

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Tras normalizar la matriz resultante sería esta:

$$\begin{bmatrix} 0.5 & 0 & 0 \\ 0.5 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

También trato los nodos sumideros en el proceso de normalización así que en realidad la matriz resultante es esta:

$$\begin{bmatrix} 0.5 & 0.33 & 0 \\ 0.5 & 0.33 & 1 \\ 0 & 0.33 & 0 \end{bmatrix}$$

Partiendo de esta matriz ya podemos calcular el PageRank:

Listing 1.2. PageRank

```
def page_rank(self, damping=0.85, limit=1.e-8):
        n = len(self.nodes)
        v = np.ones(n)
        v /= n  # Initialize v with 1/number of nodes

        error = 1
        # Iterate while mean quadratic error is greater than limit
        while error > limit:
            prev_v = v
            v = damping * np.dot(self.m, v) + (1 - damping) / n
            error = self.quadratic_error(prev_v, v)

        scores = {node: score for (node, score) in
                  zip(self.nodes, v.tolist())}
        return scores
```

Inicializo una matriz v con valores $\frac{1}{num\ nodos}$ y a partir de ahí empiezo a iterar. La formula usada para calcular cada iteracin es:

$$v_n = d * M \cdot v_{n-1} + \frac{1-d}{n}$$

La probé extensivamente y da los mismos resultados que la vista en clase (mismo orden de nodos, siempre) pero con la ventaja de que mantiene la propiedad de que la suma del PageRank de los nodos del grafo siempre es 1. d es el factor

de damping o teletransporte que por defecto es 0.85, pero se puede pasar otro por parámetro si le apetece al usuario.

Se controla la convergencia del algoritmo mediante el error cuadrático, para ello hice una pequeña función que no adjunto aquí para no marear la perdiz (veáse código). Cuando el error cuadrático es menor que el límite establecido (por parámetro, o por defecto $1e-8$)

Como resultado de la ejecución del PageRank se devuelve un diccionario nodo: puntuación

El resultado de ejecutar PageRank sobre el grafo de la Wiki que se pide en el ejercicio se puede ver en la **Figura 2**. Los nodos x son los nodos anónimos que aparecen en el grafo. El resultado coincide como se puede ver. Adjunté la representación textual del grafo en el fichero graph_wiki.txt

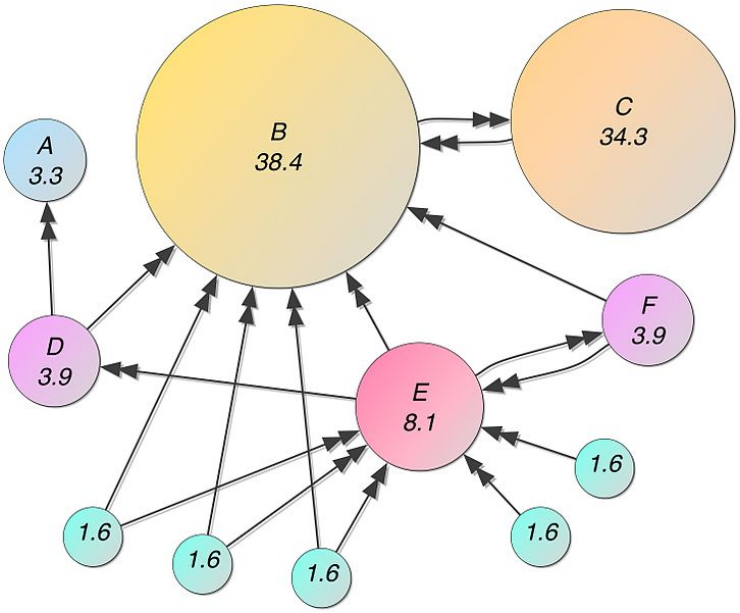

Fig. 1. El grafo de la wiki que se pide en el guión

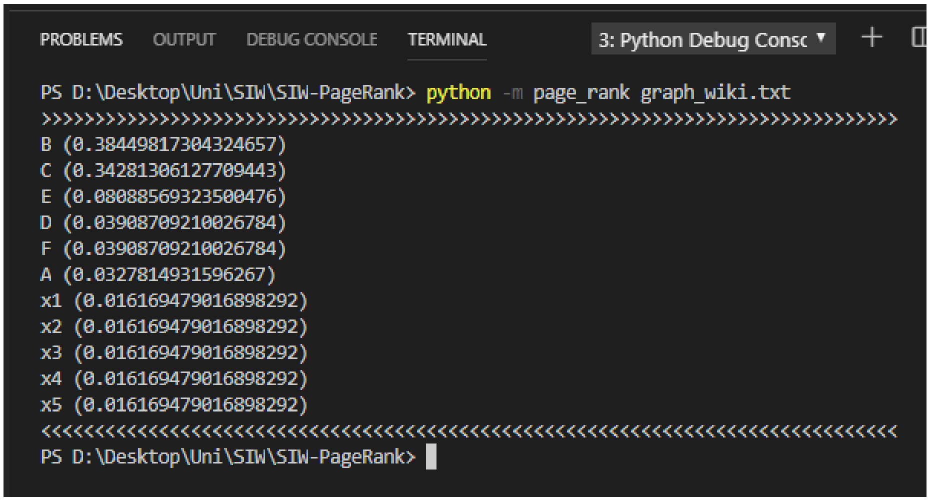

Fig. 2. PageRank del grafo contenido en graph_wiki.txt

Evidentemente se pasan todos los tests proporcionados (**Figura 3**).

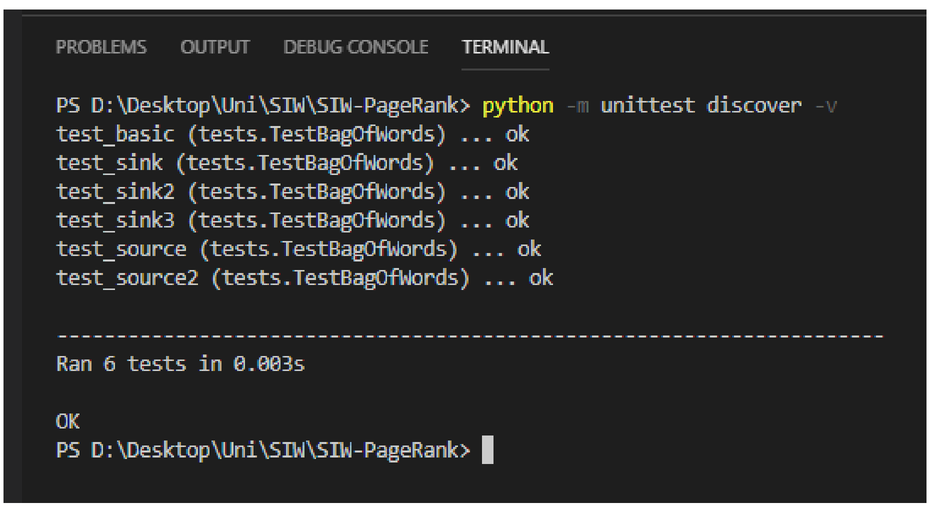

Fig. 3. PageRank del grafo contenido en graph_basic.txt

2 TextRank

Tras la lectura del artículo sobre PageRank [2] se hacen una serie de cuestiones. Lo primero que se pregunta es cómo maneja mi implementación los enlaces

salientes repetidos. Realmente los ignora, solo se considera una arista del nodo A al B. De otro modo una página podra spammear el mismo enlace muchas veces para que el grueso de su PageRank se transmitiera a un nodo en particular. Así que respondiendo a la segunda pregunta que se plantea, no parece fructífero considerar los enlaces repetidos en el contexto de la Web.

Otra cosa son los textos, en este caso podría ser interesante ponderar el peso de la relación entre un nodo y otro, como veremos, por ejemplo, en la extracción de sentencias en la que usaré métricas de similitud.

Hablaré de como extraje la sentencias más adelante.

2.1 Extracción de términos más relevantes

La implementación de TextRank se realizó muy originalmente en el fichero text_rank.py En lo que se refiere a la extracción de los términos más relevantes, se puede dividir en tres pasos:

1. Extracción de los términos del texto
2. Obtención de términos co-ocurrentes para generar las aristas del grafo
3. Aplicación de PageRank sobre el grafo

Listing 1.3. Extracción de términos

```
def extract_keywords(self, text):
        tokens = word_tokenize(text)
        # For syntax filtering, see comment further down
        tokens = pos_tag(tokens)

        keyword_list = []
        for token, tag in tokens:
            token = token.lower()
            token = token.strip(string.punctuation)
            if token and token not in string.punctuation \
                    and token not in stopwords.words("english") \
                    and tag in ["NN", "NNS", "NNP", "NNPS",
                     "JJ", "JJR", "JJS"]:
                    # Only nouns and adjectives
                    keyword_list.append(token)

        return keyword_list
```

Para extraer los términos se llevó a cabo de forma similar a como hice en las prácticas anteriores. Utilizo el tokenizador de NLTK, paso a minúsculas, elimino signos de puntuación... Esta vez prescindí de la stemmización. Las palabras vacías deben ser eliminadas pues no nos interesan aristas entre palabras vacías y verdaderos términos.

Una novedad que introduje es el filtrado sintáctico. En el artículo [2] dicen que obtuvieron mejores resultados filtrando sustantivos y adjetivos, así que empleé

NLTK para marcar los candidatos a término sintácticamente y quedarme solo con los sustantivos y adjetivos.

Una vez extraídos los términos, hay que buscar co-ocurrencias entre ellos para conseguir una lista de aristas con las que componer un grafo.

Listing 1.4. Co-ocurrencia de términos

```
def coocurrent_terms(self, terms, window=2):
        term_edges = []
        for i, word in enumerate(terms):
            for j in range(i + 1, i + window):
                if j >= len(terms):
                    break
                edge = [word, terms[j]]
                if edge not in term_edges:
                    term_edges.append(edge)
        return term_edges
```

Para esto lo que hago es establecer un rango, por defecto 2, pero se puede especificar por parámetro en línea de comandos. Si dos términos se encuentran a una distancia comprendida en ese rango, consideraremos que existe una arista entre ellos. De este modo obtenemos un grafo al que podremos aplicar un algoritmo de ranking como pueda ser PageRank

Listing 1.5. Generación del grafo y aplicación PageRank

```
def get_top_keywords(text, num_words = 10, window=2):
    tr = TextRank()
    keywords = tr.extract_keywords(text)
    edges = tr.coocurrent_terms(keywords, window)
    g = Graph(edges, undirected=True) # Using an undirected graph
    scores = g.page_rank()

    sorted_result = [(node, value) for node,
        value in sorted(scores.items(), key=lambda x: x[1], reverse=True)]
    return sorted_result[:min(len(sorted_result), num_words)
```

A partir de los ejes se construye un grafo no dirigido, ya que solo nos importa que las palabras estén conectadas entre sí. Es aquí donde se ve el por qué de aquel flag undirected en la implementación del grafo. Simplemente implica que para crear un grafo no dirigido tendré que calcular la matriz simétrica a la adyacente.

Volviendo al grano, se crea el grafo, se aplica PageRank sobre él y se devuelven los n términos especificados por el usuario con mayor ranking.

Adjunto dos textos de ejemplo en dos ficheros, metamorphosis.txt (un fragmento del libro de Kafka) y starwars.txt (un monólogo de Palpatine) El resultado de aplicar TextRank con una ventana de co-ocurrencia = 2 en ambos textos se puede ver en la **Figura 4**

```
PS D:\Desktop\Uni\SIW\SIW-PageRank> python -m text_rank metamorphosis.t
Extracting top 10 keywords
- gregor: 0.032267
- boss: 0.014131
- voice: 0.013063
- bed: 0.011972
- train: 0.011908
- little: 0.010733
- fur: 0.009134
- time: 0.008712
- door: 0.008401
- legs: 0.008247
PS D:\Desktop\Uni\SIW\SIW-PageRank> python -m text_rank starwars.txt
Extracting top 10 keywords
- dark: 0.058442
- apprentice: 0.053453
- sith: 0.049758
- force: 0.049391
- darth: 0.041178
- plagueis: 0.038276
- others: 0.035969
- side: 0.035931
- ironic: 0.032939
- thing: 0.031330
```

Fig. 4. Resultado de aplicar TextRank para la extracción de términos

2.2 Extracción de sentencias más relevantes

Para extraer las sentencias más relevantes el primer escollo a salvar es cómo dividir un texto en sentencias. El guión nos apunta al tokenizador de sentencias de NLTK [1] así que lo emplearé

Listing 1.6. Extracción de sentencias

```
def extract_sentences(self, text):
        # Using nltk sentence tokenizer to divide the text in sentences
        sentences = sent_tokenize(text)
        processed_sentences = []
        # Basic cleanup
        for sentence in sentences:
            sentence = " ".join(sentence.splitlines())
            sentence.strip()
            processed_sentences.append(sentence)
        return processed_sentences
```

La extracción de sentencias es muy sencilla con el sent_tokenize. Una vez obtenidas las limpio un poco para eliminar espacios vacíos y saltos de línea que pudieran quedar por ahí

A continuación hay que ver como podemos usar las sentencias para crear un grafo al que aplicar PageRank. Co-ocurrencia no es una opción en este caso,

hay que buscar una alternativa. Estaría muy bien tener alguna forma de conocer cuán relacionados estan dos nodos en el grafo. Tal vez, cómo de similares son?

Con una medida de similitud puedo darle un peso a las aristas del grafo de modo que dos sentencias muy similares tendrán una arista con peso mayor que dos sentencias poco parecidas.

Listing 1.7. Similitud entre nodos

```
def get_similarity_edges(self, sentences):
        sent_edges = []
        for i, sent in enumerate(sentences):
            bag1 = BagOfWords(sent)
            for j in range(len(sentences)):
                if (j == i):
                    continue
                # Bag of Words from the third assignment
                bag2 = BagOfWords(sentences[j])
                cosine_sim = coef_cosine(bag1, bag2)
                # [(node A, node B), cosine_sim(a,b)]
                edge = [(sent, sentences[j]), cosine_sim]
                if edge not in sent_edges:
                    sent_edges.append(edge)
        return sent_edges
```

Convenientemente habíamos hecho una práctica dedicada a calcular la similitud entre dos textos. En el módulo similitud aprovecho la estructura BagOfWords y los coeficientes de similitud empeleados en la práctica 3. Para generar la lista de aristas, recorro las sentencias creando BagOfWords a partir de ellas con los que calcular la similitud del coseno (cualquier otra valdría pero le que cogí cario después de tantas prácticas)

Una vez tengo las aristas creo un grafo con pesos, que no deja de ser una clase hija del grafo que usé hasta el momento para que admita pesos en las aristas en vez de usar una matriz de adyacencia.

Listing 1.8. Ejecución de PageRank sobre el grafo

```
def get_top_sentences(text, num_sentences = 10):
    tr = TextRank()
    sentences = tr.extract_sentences(text)
    sent_edges = tr.get_similarity_edges(sentences)
    g = WeightedGraph(sent_edges)
    scores = g.page_rank()

    sorted_result = [(node, value) for node,
        value in sorted(scores.items(), key=lambda x: x[1], reverse=True)]
    return sorted_result[:min(len(sorted_result), num_sentences)]
```

El resultado de aplicar TextRank para la extracción de las 10 sentencias más significativas sobre ambos textos es:

- metamorphosis.txt
  - he thought.: *0.028711*
  - Gregor!": *0.010172*
  - "Gregor, Gregor", he called, "what's wrong?": *0.007760*
  - "Getting up early all the time", he thought, "it makes you stupid.: *0.007069*
  - "Oh, God", he thought, "what a strenuous career it is that I've chosen!: *0.006575*
  - And after a short while he called again with a warning deepness in his voice: "Gregor!: *0.005751*
  - "Gregor", somebody called - it was his mother - "it's quarter to seven.: *0.005540*
  - Gregor, however, had no thought of opening the door, and instead congratulated himself for his cautious habit, acquired from his travelling, of locking all doors at night even when he was at home.: *0.005354*
  - At the other side door his sister came plaintively: "Gregor?: *0.005200*
  - Only then would he consider what to do next, as he was well aware that he would not bring his thoughts to any sensible conclusions by lying in bed.: *0.004981*
- starwars.txt
  - Darth Plagueis was a Dark Lord of the Sith, so powerful and so wise he could use the Force to influence the midichlorians to create life He had such a knowledge of the dark side that he could even keep the ones he cared about from dying.: *0.016089*
  - Did you ever hear the tragedy of Darth Plagueis The Wise?: *0.015327*
  - It's a Sith legend.: *0.015327*
  - The dark side of the Force is a pathway to many abilities some consider to be unnatural.: *0.015246*
  - He could save others from death, but not himself.: *0.015164*
  - He became so powerful the only thing he was afraid of was losing his power, which eventually, of course, he did.: *0.015082*
  - I thought not.: *0.015000*
  - Ironic.: *0.015000*
  - It's not a story the Jedi would tell you.: *0.015000*
  - Unfortunately, he taught his apprentice everything he knew, then his apprentice killed him in his sleep.: *0.015000*

2.3 Ejecución del script

El script de TextRank se ejecuta tal que as:

```
$ python -m text_rank fichero [-k 10] [-s 10] [-w 2]
```

El fichero de texto es el parámetro obligatgorio mientras que los otros son opcionales, siendo su valor por defecto el mostrado arriba

- -k o –keywords: número de top términos a extraer

- -s o –sentences: número de top sentencias a extraer
- -w o –window: tamaño de la ventana de co-ocurrencia

La ejecución imprimirá por consola los -k términos más relevantes así como las -s sentencias también más relevantes tras aplicar TextRank

References

1. https://www.nltk.org/book/ch03.html#sec-segmentation
2. Mihalcea, R., Tarau, P.: TextRank: Bringing order into text pp. 404–411 (Jul 2004), https://www.aclweb.org/anthology/W04-3252