Data Mining Recitation Notes
Week 3

Jack Rae

January 28, 2013

1 Information Retrieval

Given a set of documents, pull the (k) most similar document(s) to a given query.

1.1 Setup

Say we have D documents that we wish to select from. We represent each text document (eg. a web page) as a vector,

$$\mathbf{x_i} \in \mathbb{N}^m,\ i = 1, 2, \ldots, D \tag{1}$$

where the components of this vector x_{ij} represent the number of occurrences of word j in document i, for $j = 1, 2, \ldots, W$.

Thus our full corpus of documents can be represented by the matrix,

$$^1\mathbb{N}^{D\times W} \ni \mathbf{X} = \begin{bmatrix} \mathbf{x_1} \\ \mathbf{x_2} \\ \vdots \\ \mathbf{x_D} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1W} \\ x_{21} & x_{22} & \cdots & x_{2W} \\ \vdots & \vdots & \ddots & \vdots \\ x_{D1} & x_{D2} & \cdots & x_{DW} \end{bmatrix} \tag{2}$$

and our query text can be represented by its word counts in the vector,

$$\mathbb{N}^W \ni \mathbf{y} = (y_1, y_2, \ldots, y_W)\ .$$

Note: Representing text solely by word counts/frequencies is known as a 'bag of words' representation. It's simplistic because it disregards the order in which words

[1] The $\ni$ sign is just $\in$ reversed, it contains exactly the same meaning ($\mathbf{X}$ is an element of the set $\mathbb{N}^{D\times W}$)

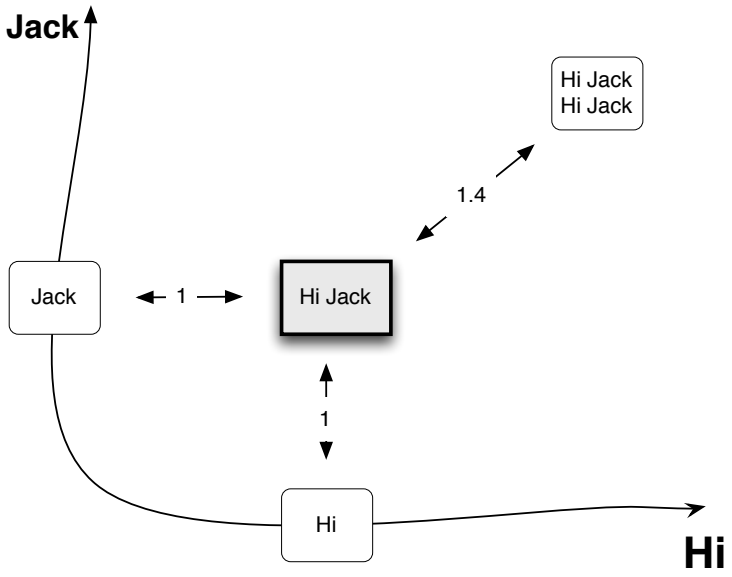
appear. We model a document as being generated from picking words out of a bag, this is never how a document is created (I hope). Nevertheless it is easy to use and often works well in practice.

1.2 Solution

We wish to find the document that is 'closest' to our query document.

Q1: What about the document that minimizes $||\mathbf{y} - \mathbf{x_i}||_2$ over $i = 1, 2, \ldots, D$? This is the main idea. We look at the distance between the two vectors, using some vector distance metric like L_2 or L_1. But one problem occurs: the **document length** affects which document is chosen. If the query is short, a long document will never be chosen even if it is highly appropriate. Consider the following example:

Figure 1: Three documents and a query: "Hi Jack", mapped into $\mathbb{N}^2$



		Document Vector	Euclidean Distance to Query
Queryt	*Hi Jack*	(1,1)	0
Document 1	Hi	(1,0)	1
Document 2	Jack	(0,1)	1
Document 3	Hi Jack Hi Jack	(2,2)	$\sqrt{2}$

With no normalization, document 3 is the 'furthest away' despite being intuitively the best match. Why? Because it's longer. That's not what we want...

Q2: Ok, so normalize by document length? That solves the problem of document length affecting our likeness measure. Letting

$$x^*_{ij} = \frac{x_{ij}}{||\mathbf{x}_i||_1}$$

translates our **word count** vector to a **word frequency** vector. We don't have to normalize by the length of the document, we could also normalize by the L_2 norm, or any norm of your choosing - as long as it's consistent with the distance metric (i.e. they match).

> **(Optional - Cosine Similarity):** Instead of minimizing the distance between points, why don't we minimize the angle between a document and a query? This technique is often referred to as **cosine similarity** and it is frequently used in scenarios involving the 'similarity' between vectors, like ours.
>
> If you inspect the first example, Figure 1, you'll see the angle between the query and the desired document ("Hi Jack Hi Jack") is 0 whereas it is $\pi/4$ for the two less desirable documents ("Hi" & "Jack"). Interestingly, minimizing the angle between two vectors is the same as minimizing the distance of normalized vectors.
>
> **Proof** Lets assume we are interested in a particular distance metric, say L_2 (Euclidean distance). We assume the vectors are normalized.[2]
>
> $$\begin{aligned}
> \underset{\mathbf{x_1},\mathbf{x_2},\ldots,\mathbf{x_n}}{\arg\min}\ angle(\mathbf{x_i}, \mathbf{y}) &= \underset{\mathbf{x_1},\mathbf{x_2},\ldots,\mathbf{x_n}}{\arg\max}\ Cos(angle(\mathbf{x_i}, y)) \\
> &= \underset{\mathbf{x_1},\mathbf{x_2},\ldots,\mathbf{x_n}}{\arg\max}\ \frac{\mathbf{x_i} \cdot \mathbf{y}}{||\mathbf{x_i}|| \times ||\mathbf{y}||} \\
> &= \underset{\mathbf{x_1},\mathbf{x_2},\ldots,\mathbf{x_n}}{\arg\max}\ \mathbf{x_i} \cdot \mathbf{y} && (Normalized) \\
> &= \underset{\mathbf{x_1},\mathbf{x_2},\ldots,\mathbf{x_n}}{\arg\min}\ 2 - \mathbf{x_i} \cdot \mathbf{y} \\
> &= \underset{\mathbf{x_1},\mathbf{x_2},\ldots,\mathbf{x_n}}{\arg\min}\ ||\mathbf{x_i}||^2 + ||\mathbf{y}||^2 - \mathbf{x_i} \cdot \mathbf{y} \\
> &= \underset{\mathbf{x_1},\mathbf{x_2},\ldots,\mathbf{x_n}}{\arg\min}\ ||\mathbf{x_i} - \mathbf{y}||^2 \\
> &= \underset{\mathbf{x_1},\mathbf{x_2},\ldots,\mathbf{x_n}}{\arg\min}\ ||\mathbf{x_i} - \mathbf{y}|| \quad \square && (By\ monotnicity)
> \end{aligned}$$
>
> **Q3: Are all words of equal interest?** When we are trying to retrieve the appropriate document, we aren't really interested in all of the words in

[2] Argmax? If $x^* = \arg\max f(\cdot)$ then x^* maximizes $f(\cdot)$

the query and all of the words in the document - in reality just a few words characterize the content of a query, and the content of a document.

Which words are useful? That is a really hard problem. A simple approach we can take, is to note that frequent words are usually less informative. This motivates **Inverse Document Frequency**: weight each frequency entry x_{ij} by

$$\log\left(\frac{D}{n_j}\right)$$

where

$$n_j = \sum_{k=1}^{D} \mathbb{1}\{\text{word j is in document k}\}$$

Q4: So to conclude? Create your word count matrix $\mathbf{X}$ and then let $\mathbf{X}^*$ be defined by

$$x^*_{ij} = \frac{x_{ij}}{||\mathbf{x_i}||} \cdot \log\left(\frac{D}{n_j}\right)$$

and choose the document which minimizes

$$||\mathbf{x_i^*} - \mathbf{y}||$$

where you can choose whatever distant metric, as long as it's consistent. Also consider **stemming** words to increase the reliability of the process, and if there is user feedback - use **Rocchio's Algorithm**.

1.3 Example: Recommendation Systems

Consider the CEO of Amazon walks up to you (in 2003) and says, "I would like to make billions of dollars by recommending my customers products at the checkout. Can you help?"

We can use the same tools from information retrieval. Say we have D previous checkout shopping carts, and W unique products, each shopping cart has a 'purchase vector' $\mathbf{x_i}$ where x_{ij} equals the number of times cart j has purchased product i. Given a sample shopping cart, $\mathbf{y}$, we can locate the (k) most similar shopping carts, and then suggest items from these carts that the customer has **not** picked up.

This is called **collaborative filtering**, here I describe a shopping cart based collaborative filtering, but you could base it around customers' full purchase history (customer based collaborative filtering) or the products themselves (item-by-item collaborative filtering). Amazon invented item-by-item collaborative

filtering in 2003 [2], these days about 40% of their sales are made from their recommendation system - so it clearly worked!

2 Pagerank

Larry Page & Sergey Brin had an original idea in the late 90s whilst studying their PhD at Stanford, integrate query relevance with web page importance when designing a search engine [1]. Namely, determine the importance of a web page by the structure of the internet, represented as a graph. This motivates the Pagerank algorithm, arguably one of the most lucrative algorithms ever created.

2.1 The Model

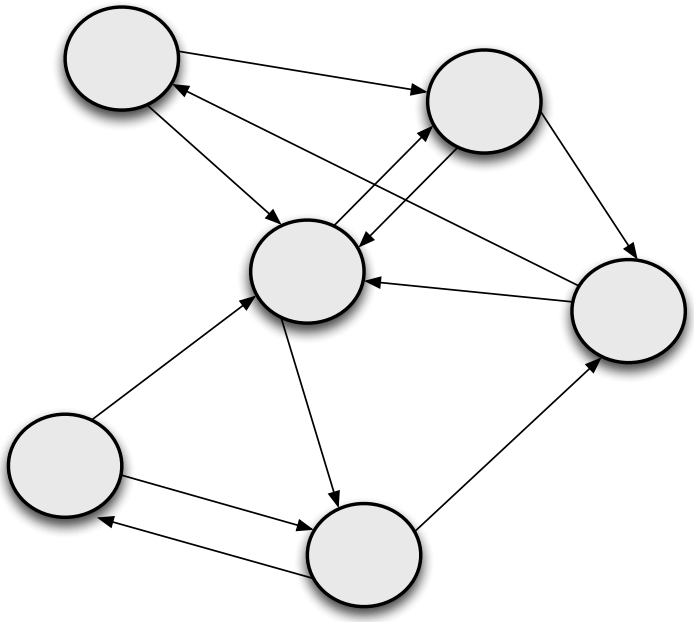
If we knew the exact number of hits every web page gets, then we could use this to determine web page importance. But this is private information that people will not want to honestly report - so Google's 20th century strategy consists of examining hyperlinks.[3] If a page is linked to by lots of other web pages, it is probably important. If a page is linked to by important web pages, it is probably important.

The key elements of the model,

- A directed graph is created from the internet. Vertices are web pages, a directed edge from $i \to j$ indicates i contains a hyperlink to j.
- We model a 'random surfer' who clicks on a random link on each page and surfs the web in this manner. With probability $1 - d$ the random surfer is picked up and dropped on a (uniformly) randomly chosen web page. This allows the random surfer to access all web pages with non-zero probability, even if the graph is not **strongly connected**.
- Assuming the internet does not form a bipartite graph, and assuming there are no sinks (web pages that have no outgoing links), the probability distribution of the position of the random surfer converges to a **stationary distribution**.
- This stationary distribution is our pagerank - a high probability (page rank) implies the random surfer would visit that particular website frequently.

[3] One might argue their 21st century strategy consists of knowing everything that happens on the internet.

Figure 2: Directed graph, the center node is frequently linked to.



The mathematics:

- n is the number of websites on the internet.
- $\{0,1\}^{n\times n} \ni \mathbf{L}$ is our graph adjacency matrix, $L_{ji} = \mathbb{1}\{\ i \text{ links to } j\ \}$.
- m_i is the number of links on page i, i.e. the out-degree of the vertex.
- $\mathbf{R}^{n\times n} \ni \mathbf{P}$ is the probability transition matrix of our random surfer,

$$P_{ji} = \frac{1-d}{n} + d \cdot \frac{L_{ji}}{m_i} \tag{3}$$

  is the probability of the random surfer moving from website $i \to j$. The columns of $\mathbf{P}$ sum to one.

- The vector of pageranks, μ is the stationary distribution of the random surfer. Namely $\mathbf{P}\mu = \mu$, so it is an eigenvector of $\mathbf{P}$ with eigenvalue 1.

We can find μ either by computing the eigenvector decomposition of $\mathbf{P}$, which requires $\mathcal{O}(n^3)$ time. Alternatively we can approximate it in $\mathcal{O}(n^2)$ using the **power iteration** method:

1. Let $\mu_0 = (1/n, 1/n, \ldots, 1/n)$.

2. Let $\mu_i = \frac{P \cdot \mu_{i-1}}{||P \cdot \mu_{i-1}||}$ $i = 1, 2, \ldots$ (i.e. iteratively multiply $\mathbf{P}$ by μ_i and normalize.

Incorporation of Pagerank Once we have our pagerank scores, we can incorporate them into our search by using them as weights in our vector similarity function, we can filter only important website that match a query, or we could sort our search results by importance.

(**Optional - Power Iteration**) You may wonder why the power iteration method works. The algorithm is a general method for finding the single eigenvector that corresponds to the largest eigenvalue - if it exists. By only computing this one eigenvector, we save a lot of computational effort. How do we know that μ corresponds to the largest eigenvector? This is obtained from the Perron-Frobenius theorem, a fundamental result in graph theory. The argument is as follows:

1. $\mathbf{P}$ is a positive matrix (i.e. all elements are positive), so the theorem is applicable.

2. As all the columns sum to one, we see there is a trivial left eigenvector corresponding to the eigenvalue of 1, the all-one row vector: $\mathbf{1} = (1, 1, \ldots, 1)$. Specifically $\mathbf{1} \cdot \mathbf{P} = \mathbf{1}$.

3. Thus, from the PF theorem, 1 is the largest left and right eigenvalue (it is called the Perron root). Why? Because the theorem states only one root is positive, and this is the Perron root.

4. Recall our pagerank vector μ is a (right) eigenvector corresponding to eigenvalue 1, this is the largest eigenvalue from the PF theorem and thus the power iteration method converges to μ.

(**Optional - Convergence Rate**) As stated in lectures, the number of iterations required for the power iteration method to converge within ϵ is dependent on two things: 1) How far μ_0 is from μ and 2) The size of λ_2, the second largest eigenvalue. The convergence is geometric with rate $1/|\lambda_2|$.

References

[1] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30(1-7):107–117, April 1998.

[2] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76 – 80, jan/feb 2003.