Algorithms for Massive Datasets

Course held by Prof. Dario Malchiodi

Federico Cristiano Bruzzone
Id. Number: 27427A

MSc in Computer Science

UNIVERSITY OF MILAN
Computer Science Department
ADAPT-Lab

Academic Year 2022–2023

COMPUTER SCIENCE DEPARTMENT • ADAPT-LAB

Contents

1 Data Mining

1.1 Things Useful to Know

1. The **TF.IDF** (*Term Frequency times Inverse Document Frequency*) measure of word importance.
2. Hash functions and their use.
3. Secondary storage (disk) its effect on running time of algorithms.
4. The base e of natural logarithm and identities involving that constant.
5. Power laws.

1.1.1 Importance of Words in Documents

Classification often starts by looking at documents, and finding the significant words in those documents. Our first guess might be that the words appearing most frequently in a document are the most significant. However, that intuition is exactly opposite of the truth. The formal measure of how concentrated into relatively few documents are the occurrences of a given word is called **TF.IDF**. It is normally computed as follows. Suppose we have a collection of N documents. Define f_{ij} to be the *frequency* of term (word) i in document j. Then, define the *term frequency* $\boldsymbol{TF}_{ij}$ to be:

$$\boldsymbol{TF}_{ij} = \frac{f_{ij}}{\max_k f_{kj}}$$

That is, the term frequency of term i in document j is f_{ij} normalized by dividing it the maximum number of occurrences of any term (perhaps excluding stop words) in the same document.

The IDF for a term is defined as follows. Suppose term i appears in n_i of the N documents in the collection. Then $\boldsymbol{IDF}_i = \log_2(N/n_i)$. The **TF.IDF** score for term i in document j is then $\boldsymbol{TF}_{ij} \times \boldsymbol{IDF}_i$.

1.1.2 Hash Functions

A hash function h takes a *hash-key* value as an argument and produces a *bucket number* as a result. The bucket number is an integer, normally in the range 0 to $B-1$, where B is the number of buckets. Hash-keys can be of any type. There is an intuitive property of hash functions that they "randomize" hash-keys.

1.1.3 Indexes

An *index* is a data structure that makes it efficient to retrieve objects given the value of one or more elements of those objects. The most common situation is one where the objects are records, and the index is on one of the fields of that record. Given a value v for that field, the index lets us retrieve all the records with value v in that field.

1.1.4 Secondary Storage

Disks are organized into *blocks*, which are the minimum units that the operating system uses to move data between main memory and disk. It takes approximately ten milliseconds to *access* and read a disk block. That delay is at least five orders of magnitude (a factor of 10^5) slower than the time taken to read a word from main memory. You can assume that a disk cannot transfer data to main memory at more than a hundred million bytes per second (100MB), no matter how that data is organized. That is not a problem when your dataset is a megabyte. But a dataset of a hundred gigabytes or a terabyte presents problems just accessing it, let alone doing anything useful with it.

1.1.5 The Base of Natural Logarithms

The constant $e = 2.7182818...$ has a number of useful special properties. In particular:

$$\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n = e$$

Some algebra lets us obtain approximations to many complex expression. Condiser $(1+\alpha)^\beta$, where α a small. We can rewirte the expression as $(1+\alpha)^{(1/\alpha)(\alpha\beta)}$. Then subsitute $\alpha = 1/n$ and $1/\alpha = n$, so we have that $(1+\frac{1}{n})^{n(\alpha\beta)}$, which is

$$\left(\left(1+\frac{1}{n}\right)^n\right)^{\alpha\beta}$$

Since α is assumed small, n is large, so the subexpression $\left(1+\frac{1}{n}\right)^n$ will be cose to the limiting value of e.

Some other usefull approximations follow from the Taylor expansion of e^x. That is,

$$\begin{aligned} e^x &= \sum_{i=0}^{\infty}\frac{x^i}{i!} \\ &= 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots \end{aligned}$$

When x is large, the above series converges slowly, although it does converge because $n!$ grows faster than x^n for any constant x.

1.1.6 Power Laws

There are many phenomena that relate two variables by a *power law*, that is, a linear relationship between the logarithms of the variables. Figure 1.1 suggest such a relationship that is: $\log_{10} y = 6 - 2\log_{10} x$

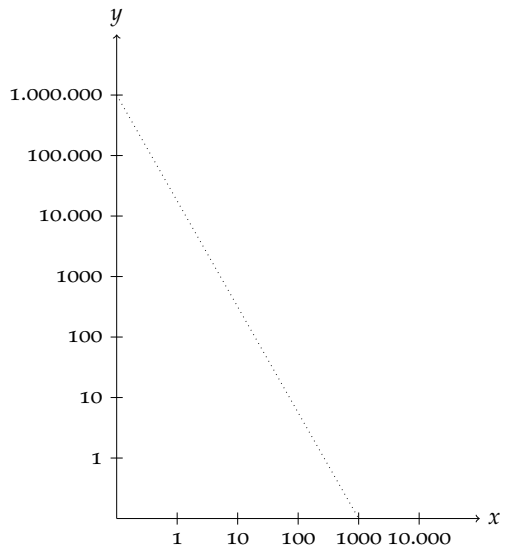

Figure 1.1. *A power law with a slope of* -2.

The Matthew Effect

Often, the existence of power laws with values of the exponent higher than 1 are explained by the *Matthew effect*. In the biblical *Book of Matthew*, there is a verse about "the rich get richer." Many phenomena exhibit this behavior, where getting a high value of some property causes that very property to increase.

2
MapReduce and Cost Model

2.1 Algorithms Using MapReduce

MapReduce is not a solution to every problem, not even every problem that profitably can use many compute nodes operating in parallel. The original purpose for which the Google implementation of MapReduce was created was executed very large matrix-verctor multiplication as are needed in the calculation of PageRank (See Chapter 3). We shall see that matrix-vector and matrix-matrix calculations fit nicely into the MapReduce style of computing. Another imortant class of operations that can use MapReduce effectively are the relational-algebra operations.

2.1.1 Matrix-Vector Multiplication by MapReduce

Suppose we have an $n \times n$ matrix M, whose element in row i and column j will be denoted m_{ij}. Suppose we also have a vector $\mathbf{v}$ of length n, whose j-th element is $\mathbf{v}_j$. Then the matrix-vector product is the vector $\mathbf{x}$ of length n, whose i-th element $\mathbf{v}_i$ is given by

$$\mathbf{x}_i = \sum_{j=1}^{n} m_{ij}\mathbf{v}_j$$

Let n be large, but not so large that vector $\mathbf{v}$ cannot fit in memory and thus be avaiable to every Map task.The matrix M and vector $\mathbf{v}$ are stored in the distributed file system (DFS). We assume that the row-column coordinates of each matrix element will be discoverable from from its position in the file, or because it is stored explicitly as a triple (i, j, m_{ij}).We also assume the position of the element $\mathbf{v}_j$ in the vector $\mathbf{v}$ is discoverable in the same way.

The Map Function: The Map function is written to apply to one element of m. Each Map task will operate on a chunk of the matrix M. From each matrix element m_{ij} it produce a key-value pair $(i, m_{ij}\mathbf{v}_j)$

The Reduce Function: The Reduce function simply sumus all the values associated with a given key i. The result will be a pair $(i, \mathbf{x}_i)$.

2.1.2 If the Vector v Cannot Fit in Memory

3
Link Analysis

Bibliography