Advances in Data Mining 2021

Wojtek Kowalczyk
w.j.kowalczyk@liacs.leidenuniv.nl

TAs: aidm@liacs.leidenuniv.nl

- *Andrius Bernatavicius*
- *Marietta Papagrigoriou*
- *Hainan Yu*
- *Max Peeperkorn*
- *Zhong Li*

Universiteit Leiden

Agenda

- ☐ Course Motivation
- ☐ Course Organization
- ☐ Course Overview
- ☐ Hash functions, hash tables, Bloom filter
- ☐ Assignment 0

Motivation behind the course

- "Classical" data mining (classification, regression, clustering) doesn't really develop anymore:
  SVM, RF, GBDT, t-SNE… **more than 10 years old**! (?)

- BigData revolution : Internet, Telecom, tablets, smartphones (tweets, emails, text, location, voice, image, video), Bioinformatics, RFIDs, Sensors, …
  => **new problems, new challenges**

- Exponential growth of the gap between available data and data processing capabilities => **need for faster algorithms, bigger/more computers**

Exponential “data flood”

- Moore’s Law: “processing speed doubles every **18** months”
- Kryder’s Law: “hard disk capacity doubles every **12** months”
- Lyman & Varian (Berkely, 2003):
  “the amount of collected data doubles every year”
  Also check:
  www.martinhilbert.net/wp-content/uploads/2015/03/HMI-Review_Hilbert2015.pdf

- Therefore every 3 years processing speed increases 4 times, while the amount of available data increases 8 times
  => the **“data flood” is doubling every 3 years!!!**

The three “V” of big data (ibm.com):

- **Volume:** Enterprises are awash with ever-growing data of all types, easily amassing terabytes—even petabytes—of information.
  - Turn 12 terabytes of Tweets created each day into improved product sentiment analysis
  - Convert 350 billion annual meter readings to better predict power consumption
- **Velocity:** Sometimes 2 minutes is too late.
  - Scrutinize 5 million trade events created each day to identify potential fraud
  - Analyze 500 million daily call detail records in real-time to predict customer churn faster
- **Variety:** Big data is any type of data - structured and unstructured data such as text, sensor data, audio, video, click streams, log files and more.
  - Monitor 100’s of live video feeds from surveillance cameras to target points of interest
  - Exploit the 80% data growth in images, video and documents to improve customer satisfaction

What do we need ?

- We need algorithms to address completely new data mining problems such as:
  - Finding similar items (photos, documents, sets/sequences of items,…)
  - Recommending items to users
  - Detecting anomalies (fraud, cyber attacks, …) in streams of transactions
  - Analyzing social networks (communities, influence, connectivity dynamics, …)
  - Detecting changing sentiment in short messages (SMS, Tweets, e-mails)
  - Discovering some patterns in (dynamic) location data
  - “Understanding” images/movies
  - …
- The algorithms should operate on huge volumes of data (or huge data streams) *on distributed systems*
- They should be fast – “time is money” - preferably real-time !!!

Textbook: http://www.mmds.org/ + Video Lectures!

Mining of Massive Datasets (Leskovec, Rajaraman, Ullman)

Chapter	Title	Book	Slides		
	Preface and Table of Contents	PDF			
Chapter 1	Data Mining	PDF		PDF	PPT
Chapter 2	Map-Reduce and the New Software Stack	PDF		PDF	PPT
Chapter 3	Finding Similar Items	PDF		PDF	PPT
Chapter 4	Mining Data Streams	PDF	Part 1: Part 2:	PDF PDF	PPT PPT
Chapter 5	Link Analysis	PDF	Part 1: Part 2:	PDF PDF	PPT PPT
Chapter 6	Frequent Itemsets	PDF		PDF	PPT
Chapter 7	Clustering	PDF		PDF	PPT
Chapter 8	Advertising on the Web	PDF		PDF	PPT
Chapter 9	Recommendation Systems	PDF	Part 1: Part 2:	PDF PDF	PPT PPT
Chapter 10	Mining Social-Network Graphs	PDF	Part 1: Part 2:	PDF PDF	PPT PPT
Chapter 11	Dimensionality Reduction	PDF		PDF	PPT
Chapter 12	Large-Scale Machine Learning	PDF	Part 1: Part 2:	PDF PDF	PPT PPT
	Index	PDF			

Course Overview

- ***Introduction*** *(1 lecture; A0 = WarmingUp Assignment)*

- ***Recommender Systems:*** *Collaborative Filtering, MatrixFactorization, ALS (2 lectures)*
- *Assignment 1: build a simple recommender system*

- ***Finding Similar Items:*** *LSH, Minhashing, Sketches (2 lectures)*
- *Assignment 2: finding pairs of similar objects*

- ***Mining Data Streams:*** *sampling, filtering, counting; on-line marketing (2 lectures)*

- ***Link Analysis:*** *PageRank Algorithm (1 lecture) Assignment 3?*

- ***Distributed Processing Massive Data: Hadoop, MapReduce, Spark*** *(2 lectures)*

- ***Optional****: SVM, RF, XGBoost, t-SNE,… (state-of-the-art since 10-20 years ;-))*

- *Anything else (on students request!)*

Course Organization

- Registration via the uSis system (done?)
- Announcements, Slides, Instructions, etc. on the Brightspace
- Course is worth 6 ECTS = 6*28h = 168h **= about 12 hours per week**
- **Three Assignments** (programming/reporting): **[60% of the final grade]**
- **Exam**: several problems, involving some formulas! **[40% of the final grade]**

To pass the course you must pass the practicals AND the exam!

Practicals

- **TA's avaliable both on-line and in Snellius labs (your preferences?)**
- **Work in couples (or solo)**, preferably Computer Science + Statistical Science
- **Presence not compulsory** (the same applies to lectures)
- **Programming (Python, NumPy, ScikitLearn, …) + Jupyter Notebook + report**
- **Assignments: solve a problem:** implement an algorithm(s), try it on some data, deliver your **code** (usually a Jupyter Notebook) and a **report** (!)
- ***The notebook should be a combination of a documented code, results, plots, tables, etc.***

Recommender Systems

- Given a few petabytes of sales data from Amazon.com in the form:

  <user_id, item_id>

  what item would you recommend to the current visitor, current_visitor_id, that is currently visiting Amazon.com and just clicked (or bought) a current_item_id?

  You have just a few milliseconds for it!

- Given a few terabytes of data collected by Netflix.com in the form:

  <user_id, movie_id, ***rating***>

  what movie would you recommend to the current visitor of Netflix.com, current_visitor_id?

  Again, you have just a few milliseconds for it!

Recommender Systems

- Recommendation algorithms can be split into:
  - Content-based: they use “features” of items or users
  - Collaborative filtering: they use data which links users to items (sales, ratings, reviews, …)
- Further, they can be split into:
  - “classical”: decision trees, logistic regression, etc.
  - distance/similarity based algorithms
  - matrix factorization based algorithms
- We will discuss a few of the most successful algorithms in depth
- Netflix Challenge: $1.000.000 for the best recommender system

Mining Data Streams

- In some situations the incoming stream of data is too fast/too big to be stored – it has to be analyzed “on-the-fly” in very limited memory…
  - Detecting fraud with electronic transactions
  - Placing “the right Ad at the right place at the right moment” in response to a click
  - Prognostic health monitoring of a fighter jet (JSF: a few TB/hour)
  - Detection of DoS Attacks
  - Answering questions like: “top 10 most frequent news items on Twitter in the last 10 seconds”, “top 10 products with the most rapidly increasing sales rate”, etc.

- A number of algorithms/building blocks have been invented:
  - Data stream sampling
  - Selective Filtering
  - Counting distinct elements in a stream
  - Real-time advertising (electronic auctions/Google Ads Machine)

Mining Data Streams:
Counting Distinct Elements

- Given a stream of IP-addresses, or user names, or documents, images, etc., answer, at any moment:
  how many distinct elements have you seen so far?

- **Naïve approach**: keep a hash table and keep track of the number of elements in the table: constant time,
  O(N) memory – might be prohibitively big!!!

- **Flajolet-Martin algorithm**: **a few bytes of memory (O(log(log(N))))** and several hash functions do the trick (approximately …)

Web Advertising: AdWords

- **Google AdWords** is Google's main advertising product and main source of revenue. Google's total advertising revenues were **USD$43 billion** in 2012 (http://en.wikipedia.org/wiki/AdWords)

- A masterpiece of: data mining, optimalisation, real-time decisioning, engineering

- How does it work?
  - Companies "offer to pay a price" for displaying their ads (per impression, per click, per banner, …) when a user enters specific words in Google search window; e.g. "Amsterdam London"
  - Usually there are several companies who compete for the same AdWords; they **bid** for these words. Not always the highest bid wins!
  - Google estimates the chance of "click-through" for all bidding parties and chooses the one which maximizes Google profit !
  - On their site, companies may change their bidding strategies; ***how should they do it?***

Finding Similar Items

- Imagine a huge collection of “digital items”, eg.:
  - Wikipedia articles
  - Photos
  - Fingerprints
  - Sales data from Amazon.com (for each user a set of items he bought)

  and a similarity measure s(o1, o2) ->[0,1]

- How much time would you need to find, for a new item, all items that are very similar to it? [We assume there are not so many such items] O(N)? or O(log(N))? **or O(1)???**

- How much time would you need to find all pairs of items with high similarity? [We assume that there are only a few such pairs]
  O(N^2)? or O(N log(N))? **or O(N)???**

- **LSH: Locality Sensitive Hashing** (random; small errors possible)

Locality Sensitive Hashing (LSH)

- A general technique for “translating” the original set of objects into a set of multiple (hundreds) of hashes, and then hashing these hashes in a special way … so the similar objects fall into the same bucket (with high probability) and non-similar objects fall into different buckets (also, with high probability)

- The technique is generic and can be used with various similarity measure, such as: *Jaccard similarity, Euclidean Distance, Hamming Distance, Cosine Distance, Edit Distance, …*

- Keywords: hash function, universal family of hash functions, random projections, minhashing, signature, randomness and probability

The PageRank Algorithm

- How Google determines the importance of a page?
- A “random surfer” model:
  - visitors start their session at random pages
  - visitors walk along links at random
  - choices are made uniformly (each outgoing link has the same chance)
  - “page importance” = “probability the surfer visits the page”
- It can be modeled by a Markov Process
  - transition matrix
  - iterative calculation of page probability distribution
  - solution = principal eigen vector of the transition matrix
- Nowadays a much more sophisticated model is used …

Implementation of PageRank

- Key ideas:
  - M is very sparse: say 10 links per page => 10 non-zeros in a column
  - keep on M on a harddisk
  - keep in RAM only the PageRank vector
  - in every iteration scan the whole matrix M, updating PageRank vector
- Some numbers:
  - n=1.000.000.000 (1 billion nodes)
  - RAM needed: 4GB (32bits per node)
  - harddisk: about 40GB (10xRAM)
  - a single scan: about 2-3 minutes
  - 50 iterations => 2-3 hours

Distributed Data Mining

- Web data sets can be very large (and distributed)
  - Tens to hundreds of terabytes (or petabytes)
- Cannot mine on a single server
- Standard architecture emerging:
  - Clusters of commodity Linux nodes
  - Gigabit ethernet interconnect (or fiberglass)
- How to handle node failures?
- How to organize computations on this architecture?

Hadoop and MapReduce

- **Hadoop Distributed File System (hdfs)**
  - chunks, replicas, “read-only”, “write/append once”
  - unlimited scalability (million of nodes)
  - very robust, can run on a cluster/grid/WAN

- **MapReduce:**
  - **Map**: “process chunks of data”
  - *shuffle and sort (implicit, pre-programmed)*
  - **Reduce**: “aggregate partial results”

Hadoop and PySpark

- Modern (Python-based) systems for processing huge data on distributed platforms

- *Distributed Data Mining at LIACS*
  - *High Performance Computing Lab:*
    Clusters with hundreds of cores, TB of RAM, hundreds of TB disk
    http://rel.liacs.nl/labs/hpclab

  - *Data Science Lab with a few big computers with*
    TBs RAM and/or many processors
    http://rel.liacs.nl/labs/dslab

Hash Functions, Hash Tables

- A *hash function* is a function which maps objects from a universe U (numbers, strings, documents, etc.) into integers {0, 1, …, N-1} (“buckets”), in a uniform way (more or less), i.e., each bucket contains more or less the same number of objects.

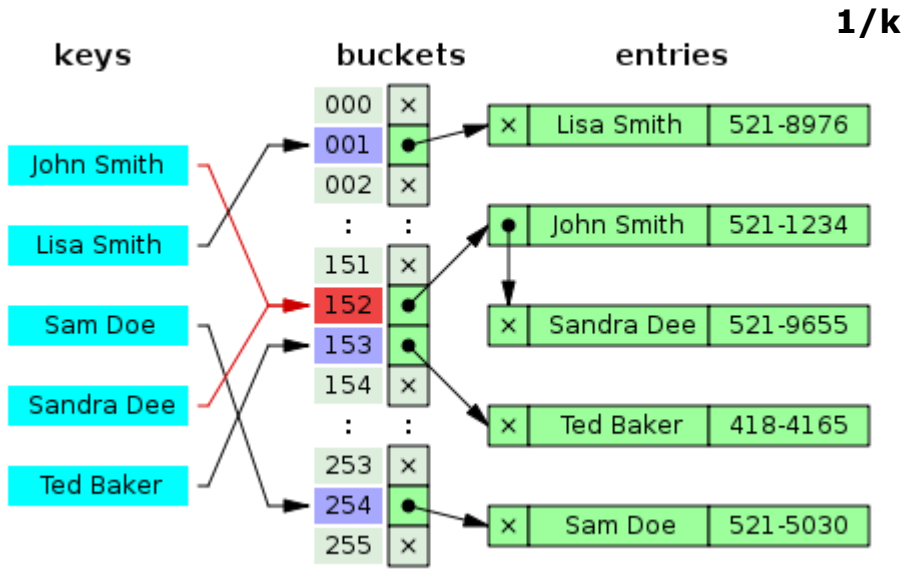
- *We will work with families of hash functions: f_1, f_2, ….*
  We assume there is a universal hash function f(i,x), such that $f_i(x)=f(i,x)$.

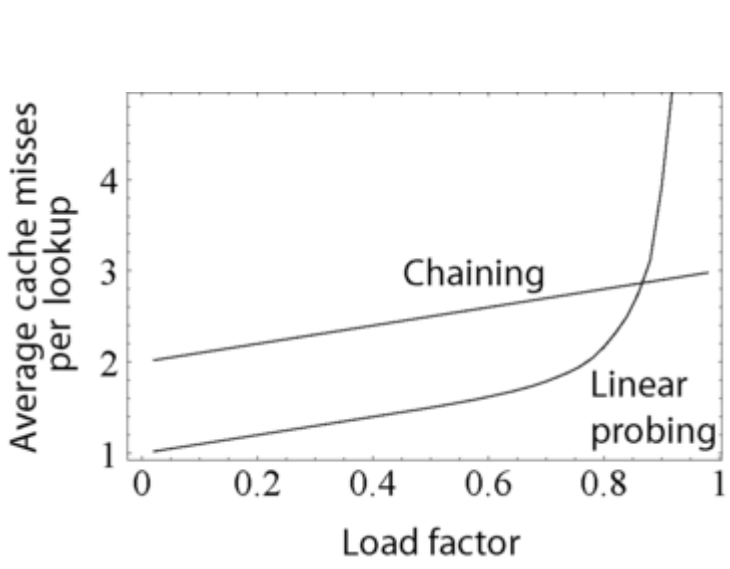
- *There is a huge body of literature on hashing. During the course we will simply assume that hash functions are available, have nice properties (“random uniformity” and “independence”) and can be computed in linear time (in the input size).*

- *To learn more check hash function and universal hashing on Wikipedia*.

Hash Tables/Dictionaries

- A *hash table* is a data structure which can store objects from a universe U, such that the expected cost of inserting, deleting and searching a single element takes O(1) steps.
- A *load factor* is the ratio K/N, where K is the size of the universe, N is the number of bins.





An important sequence

- The sequence $\mathbf{(1+1/n)^n}$ is increasing and slowly converges to $e \approx 2.7182818\ldots$
- The sequence $\mathbf{(1-1/n)^n}$ converges to $\mathbf{1/e}$
- The sequence $\mathbf{(1-1/kn)^n = (1-1/kn)^{kn/k} = ((1-1/kn)^{kn})^{1/k}}$ converges to $\mathbf{(1/e)^{1/k}}$
- **Study section 1.3.5 (The base of natural logarithms) of the textbook – we will need it frequently!**

A0: Experiments with random integers

Theory:

Suppose you generate *n* random integers between *0* and *N*;

(eg., the numpy random number generator:

r=np.random.randint(0, high=1000000, size=10000)

generates n=10.000 integers smaller than N=1.000.000)

How many *unique* values do you expect? 10.000? Less!!!

(Your random numbers might repeat!)

Find a formula that expresses this number:

unique_values(n,N) = ???

A0: Experiments with random integers

Practice:

Experiment with various values of n and N, e.g.,

n=1000, 10000;

N=n, 2n, 4n, 8n, …, 64n;

Repeat each experiment several times (with different random seeds) to get a more reliable estimates of the number of unique numbers you've generated.

Analysis:

Are the results of experiments consistent with your formula?

A0: Experiments with random integers

Deliver:

a single Jupyter notebook (in Python 3.x) that documents your theoretical considerations, experiments and conclusions.

- ***Not obligatory!***
- ***Easy deadline (2 weeks)***
- ***Reward:* feedback + grade**
  (which will not count, but will give you an idea of what we expect from you in the future)
- ***"fluency with Brightspace" (both for you and TA's)***

A0: Experiments with hash functions

Why do we play with random integers?

This is exactly what we expect from “real” hash functions: when applied to “digital objects” (e.g., strings, images, numbers, etc.) they return integer values (in a pre-specified range) that should look like random integers (e.g., two similar object should be mapped into two different integers!).

Bonus task:

Design, implement* and test any hash function that can be applied to strings (e.g., words), returning 32bit unsigned integers (uint32).

**or just find such a function somewhere!*