Before Data Preprocessing

Meng Jiang

March 25, 2014

Why “Before” Preprocessing?

- What is Data Preprocessing[1]?
- [Well studied] A large number of existing works.

S. Kotsiantis, D. Kanellopoulos, P. Pintelas, "Data Preprocessing for Supervised Leaning", International Journal of Computer Science, 2006, Vol 1 N. 2, pp 111–117.

- [Applications] An important step in data mining process and machine learning projects.
- [Importance] Analyzing data without preprocessing can produce misleading results.
- [Challenging] Take considerable amount of processing time.

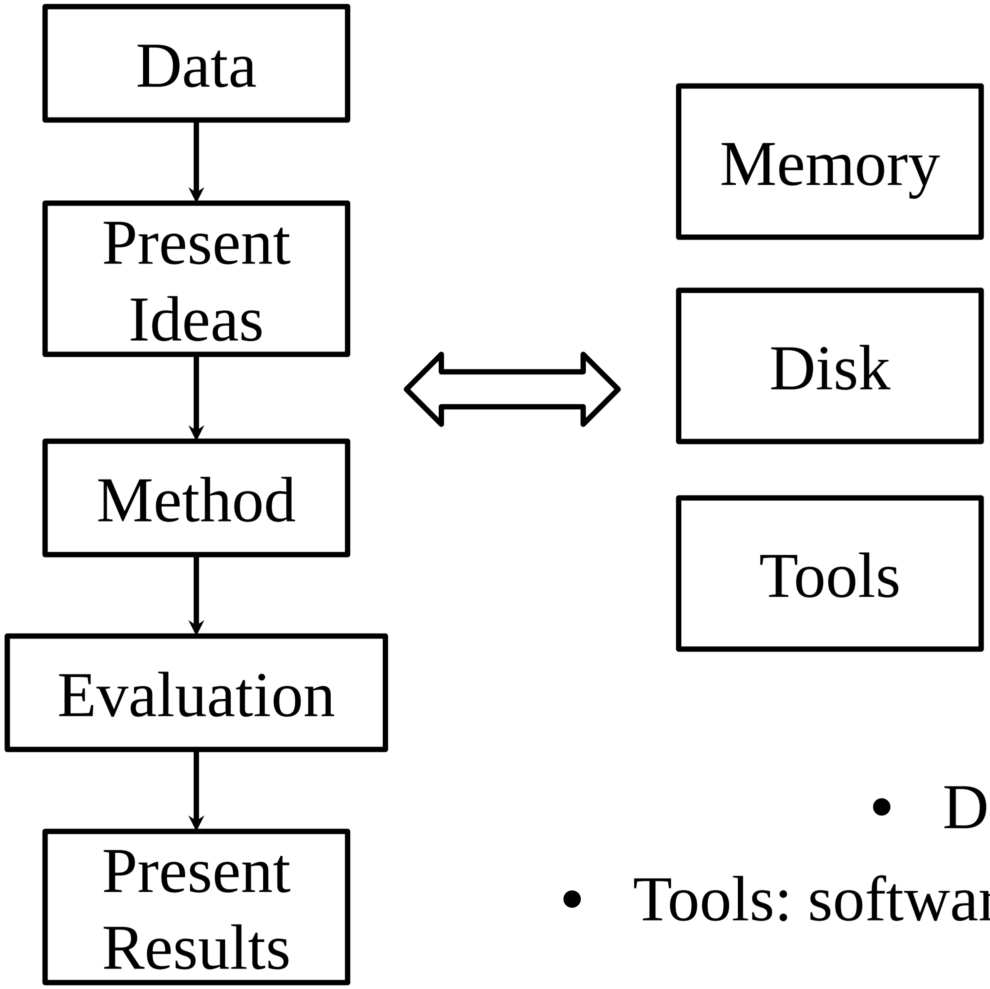
[1] http://en.wikipedia.org/wiki/Data_pre-processing

Data Preprocessing[1]

- Data cleaning
- Data integration
- Data normalization
- Data transformation
- Data reduction
- Data discretization
- Concept hierarchy generation
- Feature extraction and selection
- Here I am talking about the <u>preparation</u> for preprocessing.

[1] http://en.wikipedia.org/wiki/Data_pre-processing

Outlines



- Disk: folders, files
- Tools: software, packages, etc.

Principles

- 1. NO ERROR in DATA
  - Noisy: errors, outliers
  - Incomplete: lacking value
  - Inconsistent: age vs date of birth, sex vs pregnant
- Solutions
  - Check data types (NULL, empty; integer, enumerate, time, IP, etc.)
    - Integer: negative, float, string?
    - Time: zero?
    - IP: zero?
  - Add types for reading (using functions or other files)
    - Time: long → YYYY-MM-DD HH:MM:SS
    - IP: long → XXX.XXX.XXX.XXX
    - User: ID → nickname

Principles

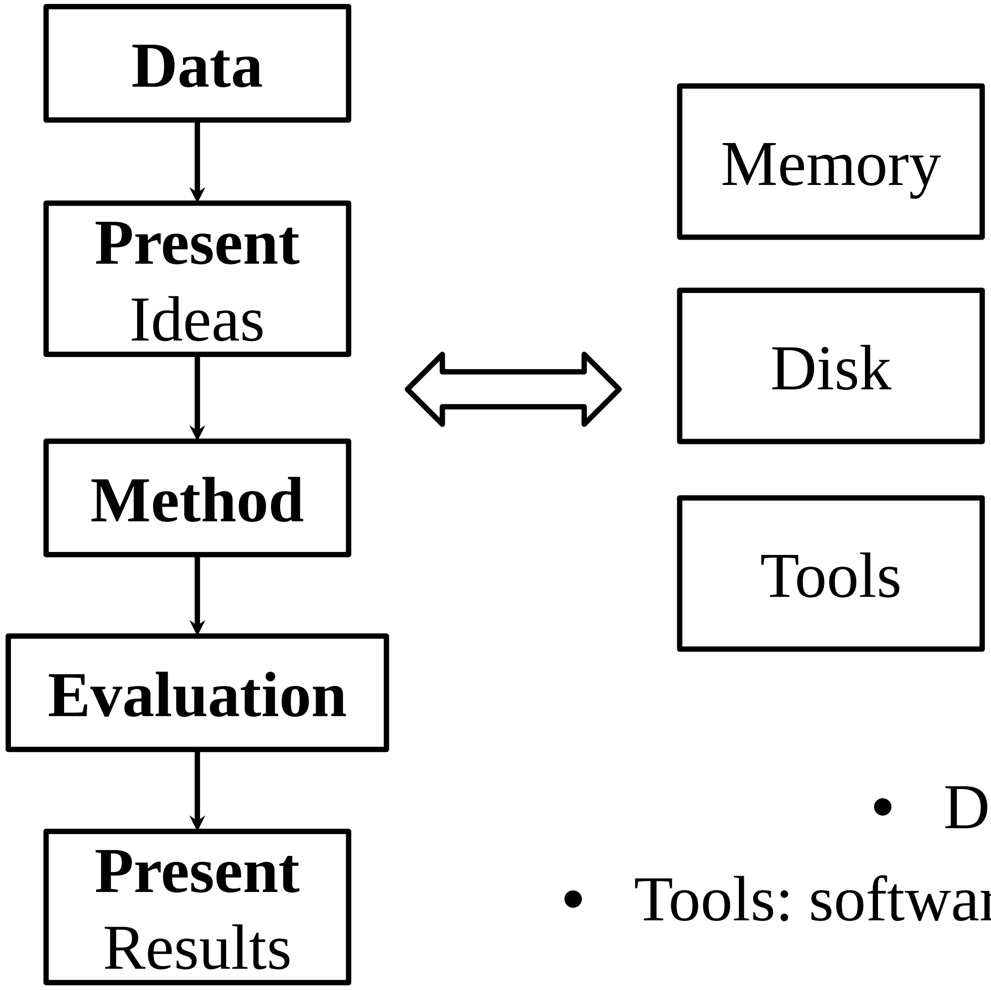
- 2. NO ERROR in CODE
  - Grammatical errors: compiling error, linked error?
  - Data: out of index, string into integer?
- Solutions
  - Debug: screen output (print, disp, etc.), file output
  - Use “break;” at the end of loops where mistakes often happen.

- 3. Readme
  - Let it be easy to understand and safe to use your data and code copies.
- Solutions
  - Plan of preprocessing in your mind; if it’s not clear, write it down.

Trade-offs

- Data
  - Data quality (density)
  - Data scale
- Hardware
  - Disk (easy to save, slow to read)
  - Memory (fast to compute, small capacity)
- Use
  - Run once (large files, but must finish in a few hours/days and never fail)
  - Run for multi-times (small files, must be fast to read files and compute)
- Code
  - Multi-tasks (click “run” and take a long rest)
  - Safety (never fail, no error, correct results)

Outlines



- Disk: folders, files
- Tools: software, packages, etc.

Data: Data Source

- From webpage
  - Crawler: http, GET, POST, cookies, WireShark
  - Parser: regular expression, pattern, XML structure, output
- From API
  - Sina Weibo: java
- From co-project
  - .tar (put together), .tar.gz (compressed), .zip, .rar
  - Split, compress, backup, readme

Data: Data Understanding

- Node (entity, etc.)

Key	Value
User	Name, sex, date of birth, social tags, etc.
Tweet	User, time, IP, content, image/video links, etc.
From 0/1?	

- Edge (behavior, connection, etc.)

Key			Value
Edge	Key1	Key2	Weight (rating, adopt/reject, etc.)
-			

- Readme

Data: Data Management (File)

- Self-<u>style</u> file format
- File type: remove comma or tab from substrings!
  - .csv: Comma (‘,’)
  - .tsv: Tab (‘\t’)
  - .txt: Space (‘ ’)
- Header: yes or no!
- Python: file reading

* How long does it take to read a 300G file line by line without doing anything?

```
fr = open(FILENAME,‘rb’) # binary!
# line = fr.readline()
for line in fr:
        arr = line.strip(‘\r\n’).split(‘,’)
        …
fr.close()
```

Data: Data Management (File)

- Python: file writing

* How long does it take to print/write a 1G file?

* How many file streams can be open at the same time?

* How large the size of file is if we want to open/PageDown it with Vim/Notepad++/UltraEdit?

```
fw = open(FILENAME,‘w’)
…
fw.write(line.strip(‘\r\n’)+ ‘\n’)
…
fw.close()
```

- Matlab: dlmwrite()
- Ending with ‘\n’: If you open the files with Notepad++, UltraEdit or Vim instead of Notepad.
- Encoding and decoding: UTF8, GBK/GB2312 (?), GB18030, etc. Notepad++: convert to UTF8/ANSI.

Data: Text Processing

- Chinese Word Segmentation
- Tools (.exe usually)
  - Stanford Word Segmenter
  - ICTCLAS (Chinese Science Academy)
  - Tsinghua NLP (Prof. Maosong Sun’s group)
  - Baidu
- Input: document per line
- Output: {word:part-of-speech}
  - ‘n’: noun
  - ‘v’: verb
  - ‘a’: adj./adv.

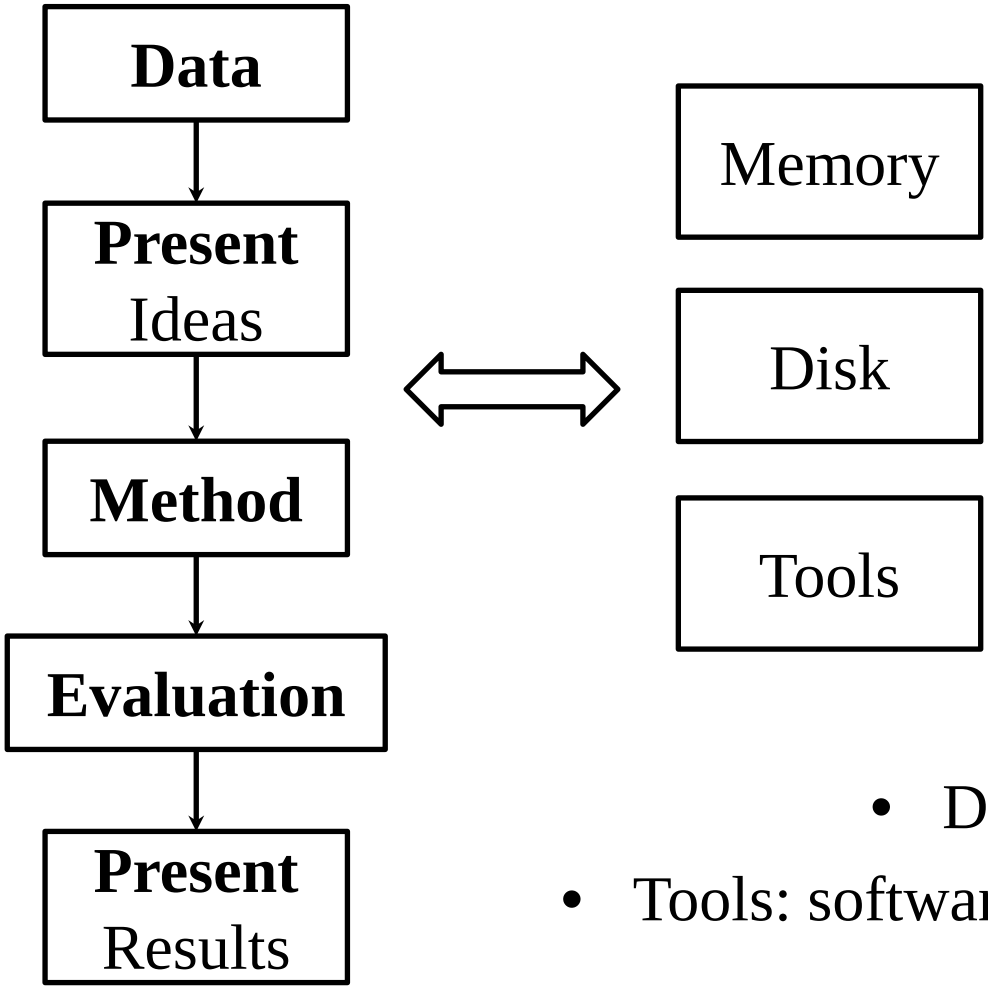
Data: Text Processing

- Word filtering
  - Document to filter words (vocabulary)
    - Part-of-speech: noun, verb?
    - Stop words
    - Word length: kill if too small
    - Word frequency: kill if too small; manually filter the top K (1000)
    - Each word must appear in at least ? (100) documents.
  - Vocabulary to filter documents
    - Each document must have at least ? (10) words from the vocabulary.
  - Until convergence

Data: Text Processing

- Advanced word filtering
  - Topic modeling tools (clustering)
    - LDA: input [document] {[word : frequency]}
    - SVD: input frequency matrix of <word, document>
  - Observe and select “topics”
  - Keep top K words in each selected topic
- Notice the data scale
  - How many words in the end? 10-100K.
  - How many documents? As many as possible.

Outlines



- Disk: folders, files
- Tools: software, packages, etc.

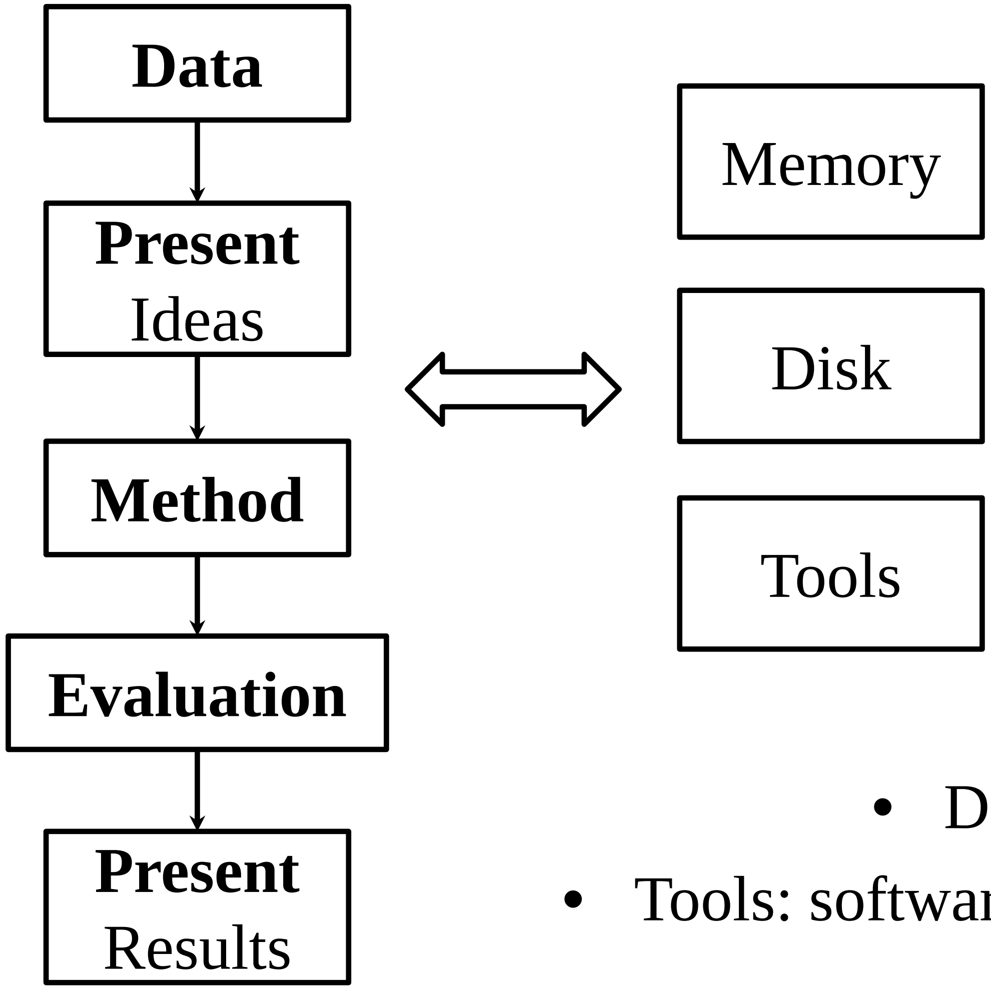
Present: Tables and Figures

- Tables

	Features, properties, labels, characteristics
Instance, object	
	Metrics (Accuracy, precision, recall, etc.)
Algorithms, techniques, models, methods	

- Figures (Matlab, R, etc.)
  - Curve plot: feature vs feature, metric vs metric (precision vs recall, etc.)
  - Scatter plot: features of nodes/edges
  - Histogram: frequency, count, comparison (evaluation), etc.
  - Heat map: an advanced presentation of scatter plot
  - 3D plot: multiple features

Outlines



- Disk: folders, files
- Tools: software, packages, etc.

Method: Data Representation

- List, map, self-defined data structure
- Quick sort – use “sort” packages and libraries
  - List: sort by which item?
  - Map: sort by key, value or item in value?
  - Sort function: never be complicated; reverse: *(-1)
  - Python:
    - Sort map XXX by key/value

```
sort_XXX = sorted(XXX.items(),key=lambda x:x[0])
sort_XXX = sorted(XXX.items(),key=lambda x:x[1])
```

    - Sort list XXX by 2nd column

```
sort_XXX = sorted(XXX,key=lambda x:x[1])
```

- Scan – careful design of your lists and maps
  - Re-ordering instances using list (new ID to ID) and map (ID to new ID)

    ID → new ID (starting from 0/1) for plots and computing

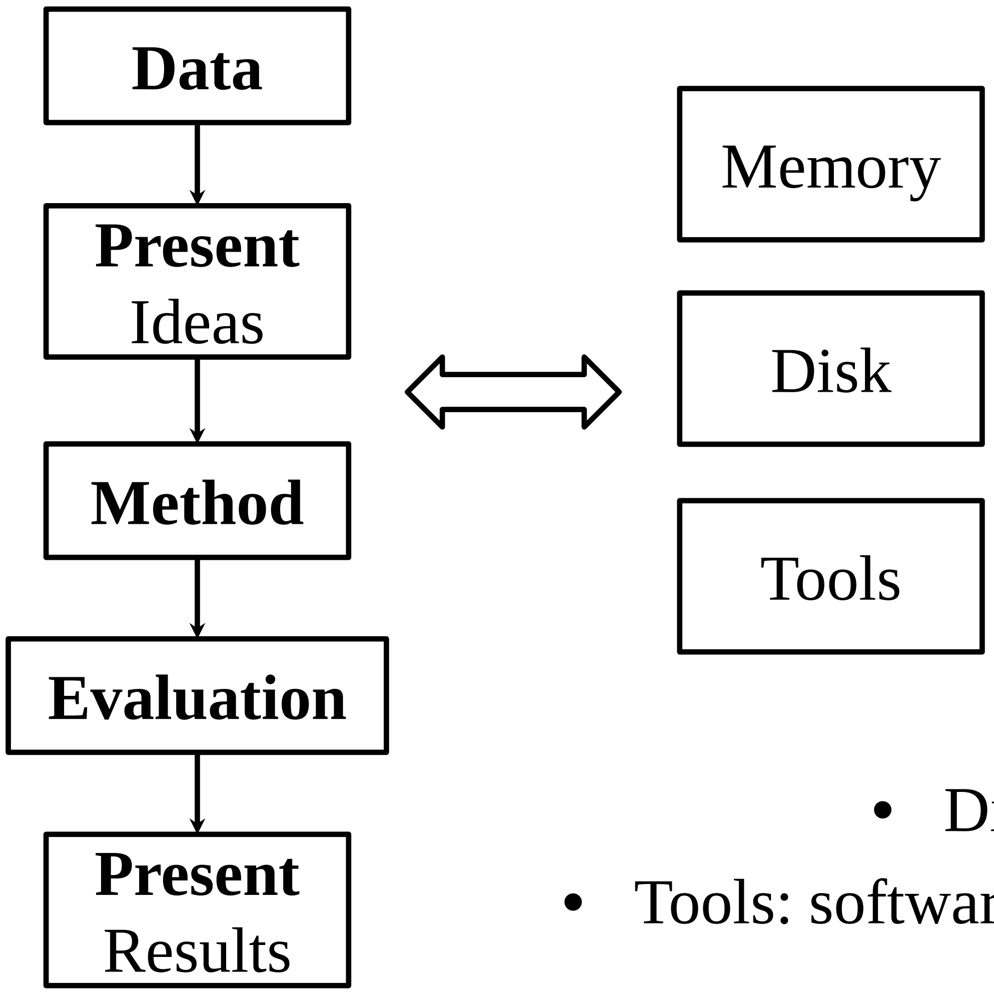
Method: Disk and Memory Use

- Re-ordering
  - R, Matlab: start from 1
  - Python (NumPy, SciPy): start from 0
- How large?
- When do what?

Disk	**Memory**	
500G/1T	4G/8G	Laptop, 32-bit system ☹
2T	**16G**	**Best choice (Ideas and experiments)**
	64G	**Paper (Experiments)**
	160G	**Run once (Preprocessing)**

- Use processes to simulate local “Hadoop”
  - Split and Combination: Map and Reduce
    - Edge list to adjacency list
    - HITS algorithm, belief propagation

Outlines

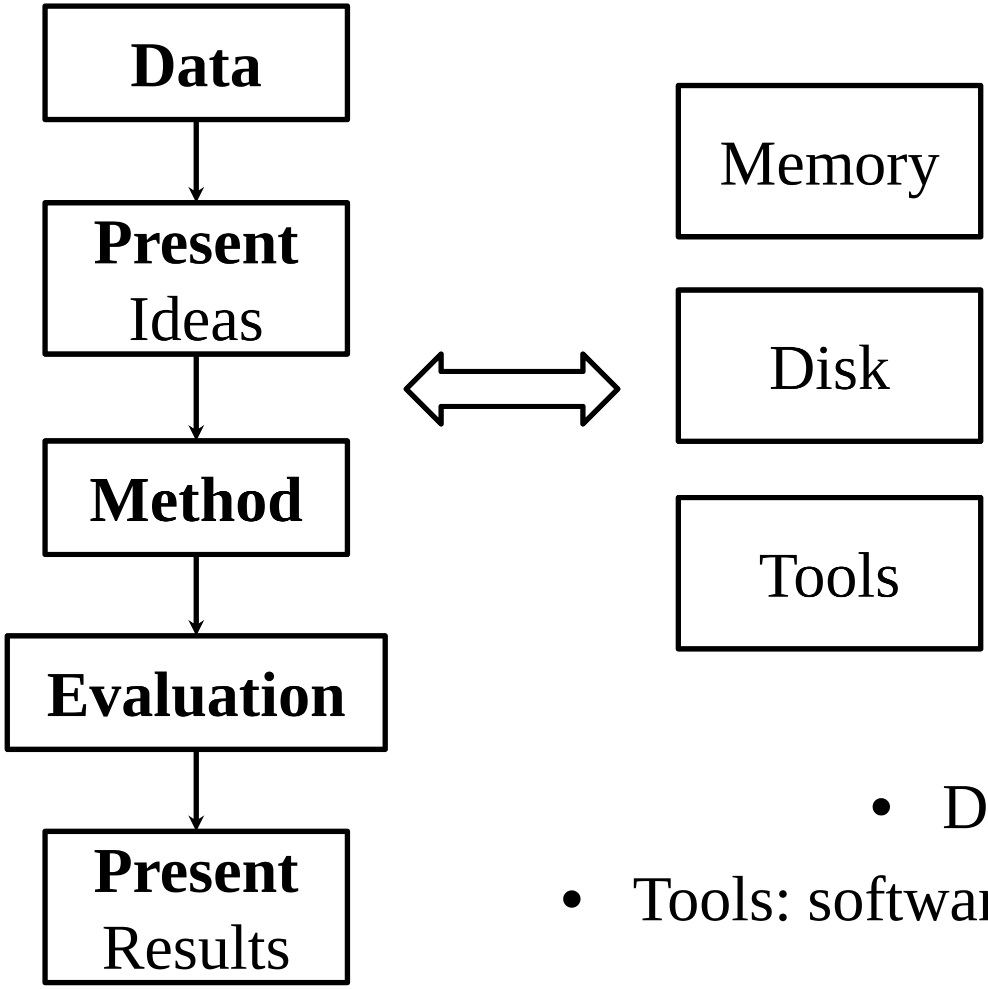


- Disk: folders, files
- Tools: software, packages, etc.

Method: Algorithm Design

- Matrix factorization
  - Matlab: Dot product (.*), multiplication (*), transform (’)
  - Loops (row or column): if N > 1K, N*N will be a disaster
    - Only one loop
    - Slow to get entries from the matrix
  - Step length for gradient descent
    - Small (converge slowly)
    - Big (converge fast, but often fail)
    - Fix or adjust the number
- Random walk with restarts
  - Normalization

Outlines



- Disk: folders, files
- Tools: software, packages, etc.

Evaluation: Metrics

- Error-based
  - MAE
  - RMSE
- Top K
  - HITS, accuracy
- Prediction
  - TP, FP, TN, FN
  - Accuracy, AUC
  - Precision, Recall, ROC
- Ranking
  - Kendall and Spearman

End: Trade-offs when You are Doing Data Preprocessing

- ?

End: Trade-offs when You are Doing Data Preprocessing

- Data
  - Data quality (density)
  - Data scale
- Hardware
  - Disk (easy to save, slow to read)
  - Memory (fast to compute, small capacity)
- Use
  - Run once (large files, but must finish in a few hours/days and never fail)
  - Run for multi-times (small files, must be fast to read files and compute)
- Code
  - Multi-tasks (click “run” and take a long rest)
  - Safety (never fail, no error, correct results)