# Code optimization for sustainable data mining

Zalán Bodó

Faculty of Mathematics and Computer Science
Babeş–Bolyai University, Cluj-Napoca


SusTrainable Summer School, July 10–14, 2023 @ University of Coimbra, Portugal

# *Presentation outline*

sustainable coding/programming:

**interpreted languages**
(fast coding, easy debugging)
**+**
**optimized representation**
**+**
**vectorization**
(parallel execution of multiple operations)

# High-level programming languages

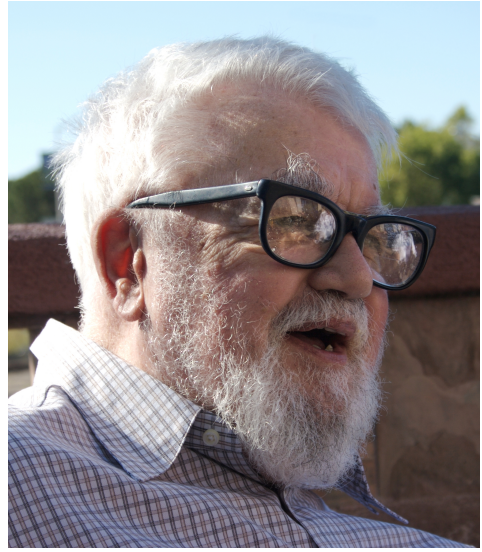
- need to write programs easier:
  - human-readable = simpler, testable/debuggable, …
- no theory
- 1957: **Fortran** – John Backus and IBM
- 1958: **ALGOL** – John Backus, Peter Naur, John McCarthy, …
- 1959: **COBOL**
- 1960: **Lisp** – John McCarthy, Steve Russel, …




# John McCarthy

– the link between programming theory and AI –

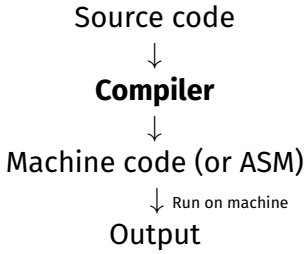
- September 4, 1927 – October 24, 2011
- one of the founders of AI
- 1956: coined the term "**artificial intelligence**" in a proposal – together with Marvin Minsky, Nathaniel Rochester and Claude E. Shannon
- Lisp, ALGOL 60
- invented *garbage collection*
- developed the first time-sharing systems – with others

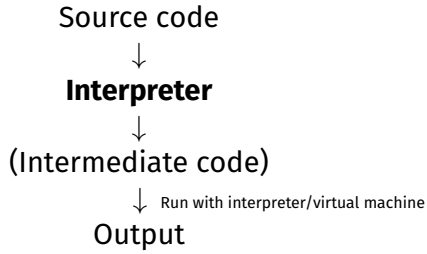


# The inside of a compiler

- *lexical* analysis
  - = tokenization
  - `int x = 0;` ⇒ `int` `x` `=` `0` `;`
- *syntactic* analysis
  - are tokens in the correct order?
  - `=` `x` `int` `;` `0` OR `int` `x` `=` `0` `;`
- *semantic* analysis
  - is it meaningful?
  - the type of variable `x` and constant `0` are the same?
- code generation
  - ⇒ machine / assembly / intermediate (byte) code
- code optimization

## Compilers vs interpreters



**Compilers**

- fast code
- *rigid*: need to compile the entire code to run

⇒ more difficult to debug

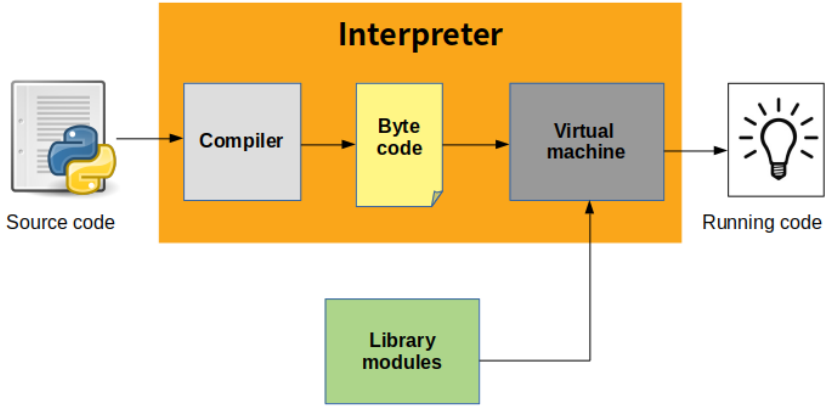
⇒ more difficult to code

**Interpreters**

- slower – because of the VM
- *flexible*: can be easily run only a small part of the code

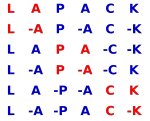
⇒ easy to debug

⇒ easy to program




From source code to running code in Python (from https://indianpythonista.wordpress.com/2018/01/04/how-python-runs/)

# Numerical libraries




- **BLAS** (Basic Linear Algebra Subprograms)[1]
  - 1979
  - Fortran and/or C
  - low-level linear algebra operations (vector, matrix additions/multiplications, dot products, etc.)
- **LAPACK** (Linear Algebra Package)[2] – based on BLAS
  - 1992
  - Fortran 90
  - higher-level linear algebra operations (solving systems of linear equations, eigenvalue, singular value problems, matrix factorizations, etc.)
- (almost) all high-level languages – having such support – use the above two libraries or their descendants (e.g. MATLAB, Python's NumPy/SciPy, Julia, R, ...)

[1] https://netlib.org/blas/

[2] https://netlib.org/lapack/

# Vectorization


- Silicon Graphics: 1985 – **MIPS** (Microprocessor without Interlocked Pipelined Stages) **RISC** (reduced instruction set computer) CPU
  - 1996 → 1999: **MIPS V** → **MIPS64**
- Intel:
  - 1997: **MMX** (MultiMedia eXtension / Multiple Math eXtension / Matrix Math eXtension) – Pentium P5
  - 1999: **SSE** (Streaming SIMD Extensions) – Pentium III
  - 2008: **AVX** (Advanced Vector Extensions) – Sandy Bridge (2011) ; later: AVX2 (2013, Haswell), AVX-512 (2016–2017, Knights Landing co-processor, Skylake)
- vector operations for vector registers – the same math operation can be performed simultaneously on multiple values
- machine-dependent *optimized* BLAS libraries can be used


- Nvidia:
  - 2007: **CUDA** (Compute Unified Device Architecture)

- vectorization
  - numerical libraries for scientific computing use architecture-specific **low-level CPU-optimized code** to execute more operations at the same time – e.g. AVX
  - vectorization can be also beneficial without CPU optimization because of using **optimized algorithms** – e.g. Strassen's algorithm for matrix multiplication which reduces $O(n^3)$ to $\approx O(n^{2.807})$
- in the following: **vectorization** = using vectors, matrices, tensors in algebraic operations

## Example

- let us suppose a Euclidean distance matrix is required:

$$D_{i,j} = \|\mathbf{x}_i - \mathbf{x}_j\|_2^2, \quad i,j \in \{1,2,\ldots,N\}$$

- calculate using matrix multiplications + additions:

$$\begin{aligned}
\mathbf{A} &= \mathbf{X}\mathbf{X}^T \\
\mathbf{B} &= diag(diag(\mathbf{A})) \\
\mathbf{D} &= -2\mathbf{X}\mathbf{X}^T + \mathbf{B}\mathbf{1}_{NN} + \mathbf{1}_{NN}\mathbf{B}
\end{aligned}$$

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix}$$

# Data mining (DM) & machine learning (ML)

- **data mining**:
  - discovering patterns in data
- **machine learning**:
  - supervised learning (classification, regression)
  - unsupervised learning (clustering, density estimation)
  - reinforcement learning

# Examples


- semantic segmentation of images
- spam filtering
- ranking in information retrieval/search engines
- article/movie/music/...recommendation
- ...

# PageRank

Google

information retrieval

Toate | Imagini | Cărți | Videoclipuri | Știri | Mai multe | Instrumente

Aproximativ 1.540.000.000 rezultate (0.41 secunde)

Information retrieval (IR) in computing and information science is the process of obtaining information system resources that are relevant to an information need from a collection of those resources. Searches can be based on full-text or other content-based indexing.


wikipedia.org
https://en.wikipedia.org › wiki › Information_retrieval

Information retrieval - Wikipedia

Despre fragmentele recomandate • Feedback

geeksforgeeks.org
https://www.geeksforgeeks.or... · Traducerea acestei pagini

What is Information Retrieval? - GeeksforGeeks

3 iul. 2022 — **Information Retrieval** is the activity of obtaining material that can usually be documented on an unstructured nature i.e. usually text which ...

stanford.edu
https://nlp.stanford.edu › inf... · Traducerea acestei pagini

Introduction to Information Retrieval - Stanford NLP Group

The book aims to provide a modern approach to **information retrieval** from a computer science perspective. It is based on a course we have been teaching in ...

tutorialspoint.com
https://www.tutorialspoint.com ... · Traducerea acestei pagini

NLP - Information Retrieval - Tutorialspoint

**Information retrieval** (IR) may be defined as a software program that deals with the organization, storage, retrieval and evaluation of information from ...

coveo.com
https://www.coveo.com › blog · Traducerea acestei pagini

What Is Information Retrieval? - Coveo

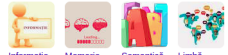
recuperarea informațiilor

Domeniu de studii

Tradus din engleză - Recuperarea informațiilor în informatică și știința informației este procesul de obținere a resurselor sistemului informațional care sunt relevante pentru o nevoie de informații dintr-o colecție de acele resurse. Căutările se pot baza pe text integral sau pe alte indexări bazate pe conținut. Wikipedia (Engleză)

Vezi descrierea inițială

Utilizatorii caută și — Vezi încă peste 10




Feedback

- Page, L., Brin, S., Motwani, R., Winograd, T. The PageRank citation ranking: Bringing order to the web. Stanford InfoLab. 1999.[3]

$$PR(p_i) = \sum_{p_j \in N^{-1}(p_i)} \frac{PR(p_j)}{|N(p_j)|}$$

where $N(x)$ and $N^{-1}(x)$ are the forward and backward neighbors of $x$, respectively

[3] They did not *invent* the algorithm:
Edmund Landau. Zur relativen Wertbemessung der Turnierresultate. Deutsches Wochenschach. 11 (42): 51–54, 1895.
Gabriel Pinski, Francis Narin. Citation influence for journal aggregates of scientific publications: Theory, with application to the literature of physics. Information Processing & Management. 12 (5): 297–312, 1976.

- problem: if there is a group/cluster with in-links only, it will “steal” the ranks from the other nodes
- solution: introducing a “teleportation” probability (modeling a random jump in the browsing process)

$$PR(p_i) = (1 - \alpha) \sum_{p_j \in N^{-1}(p_i)} \frac{PR(p_j)}{|N(p_j)|} + \frac{\alpha}{n}$$

where $n$ denotes the total number of nodes (pages)

- recursive formula:

$$\mathbf{r} = (1-\alpha)\mathbf{A}\mathbf{r} + (\alpha/n)\mathbf{1}_n$$

  where $\mathbf{A} = \mathbf{B}^T$
  - $B_{ij} = \begin{cases} 1/|N(p_i)|, & \text{ha } \exists i \to j \\ 0, & \text{otherwise} \end{cases}$
  - $\mathbf{1}_n \in \mathbb{R}^n$
- explicit formula:

$$\mathbf{r} = (\mathbf{I} - (1-\alpha)\mathbf{A})^{-1}(\alpha/n)\mathbf{1}_n$$

*Why's the $n$ in the denominator?*

- in the previous expression $sum(\mathbf{r}) = 1$
- without $n$ in the denominator the sum would be $n$ instead of $1$:

$$\begin{aligned}
\mathbf{r} &= (1-\alpha)\mathbf{A}\mathbf{r} + \alpha\mathbf{1}_n \\
\mathbf{r} &= \begin{pmatrix} (1-\alpha)(p_{11}r_1 + p_{21}r_2 + p_{31}r_3) + \alpha \\ (1-\alpha)(p_{12}r_1 + p_{22}r_2 + p_{32}r_3) + \alpha \\ (1-\alpha)(p_{13}r_1 + p_{23}r_2 + p_{33}r_3) + \alpha \end{pmatrix} \\
sum(\mathbf{r}) &= (1-\alpha)sum(\mathbf{r}) + n \cdot \alpha \\
sum(\mathbf{r}) &= n
\end{aligned}$$

- using $\alpha/n$ instead of $\alpha$, the sum of the ranks will equal $1$

`timeit`

| Vectorized | Using loops |
|---|---|
| $9.73^{-4} \pm 2.0255^{-4}$ | $2.1311 \pm 1.7474^{-2}$ |

- random graph: 1000 nodes
- iterative version: 10 iterations
- 10 runs (using the same random graph)

Using eigenvectors:

- let $\mathbf{M} = (1-\alpha)\mathbf{A} + (\alpha/n)\mathbf{1}_{nn}$
- solve the $\mathbf{r} = \mathbf{M}\mathbf{r}$ eigenvalue equation; dividing the solution by $sum(\mathbf{r})$ we obtain the ranks $\mathbf{r}$

$$\mathbf{r} = (1-\alpha)\mathbf{A}\mathbf{r} + (\alpha/n)\mathbf{1}_{nn}\mathbf{r}$$

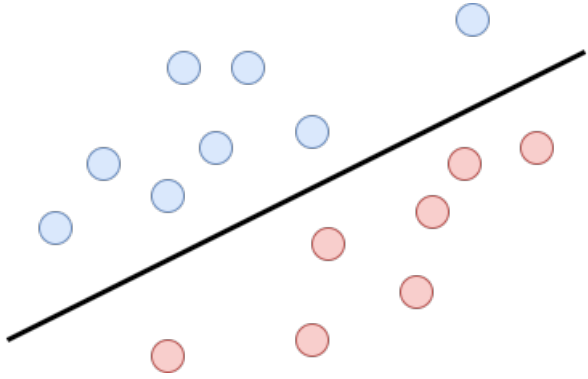
- since $\mathbf{1}_{nn}\mathbf{r} = \mathbf{1}_n sum(\mathbf{r})$, we obtain

$$\mathbf{r} = (1-\alpha)\mathbf{A}\mathbf{r} + (\alpha/n)\mathbf{1}_n sum(\mathbf{r})$$

- because $sum(\mathbf{r}) = 1$ (in the *original* formulation) $\Rightarrow$

$$\mathbf{r} = (1-\alpha)\mathbf{A}\mathbf{r} + (\alpha/n)\mathbf{1}_n$$

# Regularized least squares classification



- aka *linear regression*
- classification: binary, $\mathcal{Y} = \{-1, +1\}$
  1. $\mathbf{w}^T\mathbf{x} + b < 0 \Rightarrow -1$
  2. $\mathbf{w}^T\mathbf{x} + b \geq 0 \Rightarrow +1$

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = \begin{pmatrix} w_1 & w_2 & \dots & w_d \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix} + b = 0$$

  is the generalized equation for a line in $d$ dimensions (= hyperplane)
- we want to find $\hat{f} : \mathcal{X} \subseteq \mathbb{R} \to \mathcal{Y}$ that best approximates the unknown $f$

- data: $\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix}$, $\mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$
- problem: minimize the squared error:

$$\begin{aligned} \sum_{i=1}^{N}(f(\mathbf{x}_i) - y_i)^2 &= \sum_i (\mathbf{w}^T\mathbf{x}_i + b - y_i)^2 \\ &= \|\mathbf{X}\mathbf{w} - \mathbf{Y}\|^2 \\ &= (\mathbf{X}\mathbf{w} - \mathbf{Y})^T(\mathbf{X}\mathbf{w} - \mathbf{Y}) \\ &= \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{w}^T\mathbf{X}^T\mathbf{Y} + \mathbf{Y}^T\mathbf{Y} \end{aligned}$$

(*Note:* $b$ was embedded in $\mathbf{w}$: data was extended by a constant 1 dimension, and similarly the number of parameters in $\mathbf{w}$ was extended by one)

- differentiating by $\mathbf{w}^T$ and setting equal to zero:

$$\begin{aligned} 2\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{X}^T\mathbf{Y} &= 0 \\ \mathbf{w} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} \end{aligned}$$

- problem: $\mathbf{X}^T\mathbf{X}$ is not always invertible
- solution: introduce regularization by $\lambda > 0$

$$\min_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{Y}\|^2 + \lambda\|\mathbf{w}\|^2$$

- result:

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y}$$

  - a $\mathbf{Z} + \lambda\mathbf{I}$ matrix is always invertible, if $\mathbf{Z}$ is symmetric and $\lambda > 0$
  - size of matrix to be inverted: $d \times d$ (because $\mathbf{X} \in \mathbb{R}^{N \times d}$)
  - if $d \gg N \Rightarrow$ next slide

## Searle's formula:[4]

$$(\mathbf{A} + \mathbf{B}\mathbf{B}^T)^{-1}\mathbf{B} = \mathbf{A}^{-1}\mathbf{B}(\mathbf{I} + \mathbf{B}^T\mathbf{A}^{-1}\mathbf{B})^{-1}$$

- rewrite the solution for $\mathbf{w}$ using $\mathbf{A} := \lambda\mathbf{I}, \mathbf{B} := \mathbf{X}^T$:

$$\begin{aligned}
\mathbf{w} &= (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y} \\
&= \frac{1}{\lambda}\mathbf{X}^T(\mathbf{I} + \frac{1}{\lambda}\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{Y} \\
&= \mathbf{X}^T(\lambda\mathbf{I} + \mathbf{X}\mathbf{X}^T)^{-1}\mathbf{Y}
\end{aligned}$$

- size of the matrix to be inverted: $N \times N$

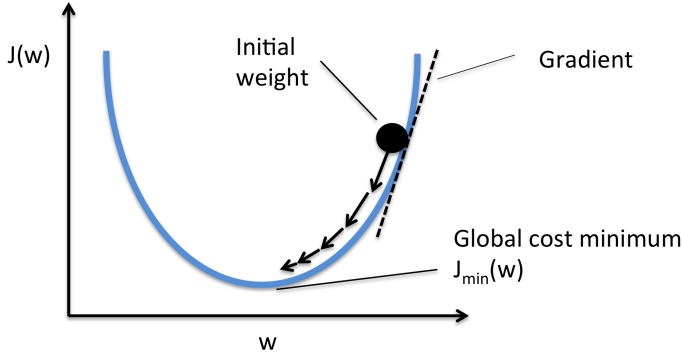
[4] K. B. Petersen, M. S. Pedersen. The Matrix Cookbook. 2012.
https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf

- **inversion** is not always possible – in the case of large matrices
- how to circumvent it?
  (1) in this case $\mathbf{w}$ equals $\mathbf{A}^{-1} \cdot \mathbf{b}$
  $\Rightarrow$ solve the system of linear equations $\mathbf{A} \cdot \mathbf{w} = \mathbf{b}$
  (2) use gradient descent to determine $\mathbf{w}$

- function to be optimized: $f : \mathbb{R}^d \to \mathbb{R}$
- gradient descent:
  - maximization: $x_{n+1} = x_n + \eta \nabla f(x_n)$
  - minimization: $x_{n+1} = x_n - \eta \nabla f(x_n)$ (opposite direction)
- the gradient always shows the direction of the maximum; why?
- answer: definition of derivative

$$\nabla f(x) = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

  if $f(x)$ is increasing $\Rightarrow \nabla f(x)$ is positive; if decreasing $\Rightarrow$ negative
- e.g. $f(x) = x^2 - 1$, $\nabla f(2) = 4 \Rightarrow$ the $\nearrow$ direction is $+\infty$;
  similarly $\nabla f(-2) = -4 \Rightarrow$ the $\nearrow$ (or more precisely $\nwarrow$) direction is $-\infty$

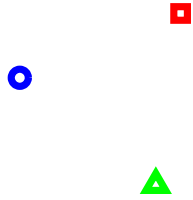



Gradient descent (from https://sebastianraschka.com/faq/docs/gradient-optimization.html)

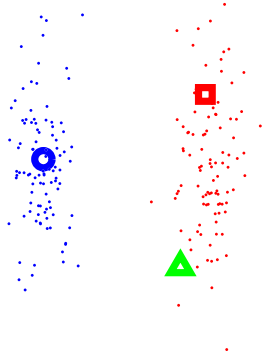
- for RLS the gradient of the error function w.r.t. $\mathbf{w}$ is:

$$\frac{1}{N}\left(\mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{Y} + \lambda\mathbf{w}\right)$$

- semi-supervised learning (SSL):



- semi-supervised learning (SSL):



- $\mathcal{D} = \{(\mathbf{x}_i, y_i)\} \cup \mathbf{X}_u$, $N = \ell + u, \ell \ll u$
- additional regularization:

$$\sum_{i,j=1}^{N} s_{ij}(f_i - f_j)^2$$

  where
  - $f_i = \mathbf{w}^T \mathbf{x}_i$ is the prediction output for $\mathbf{x}_i$
  - $s_{ij}$ denotes some similarity values between $\mathbf{x}_i$ and $\mathbf{x}_j$ (e.g. Gaussian, $\exp\left(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2\right)$)
- idea: include the condition that for similar inputs the output should be similar as well (SSL *smoothness* / *cluster* assumption)

$$\arg\min_{\mathbf{w}} \frac{1}{\ell}\|\mathbf{X}_\ell \mathbf{w} - \mathbf{Y}\|^2 + \lambda \mathbf{w}^T \mathbf{w} + \frac{\mu}{2N^2} \sum_{i,j=1}^{N} s_{ij}(f_i - f_j)^2$$

$$\Rightarrow$$

$$\arg\min_{\mathbf{w}} \frac{1}{\ell}\|\mathbf{X}_\ell \mathbf{w} - \mathbf{Y}\|^2 + \lambda \mathbf{w}^T \mathbf{w} + \frac{\mu}{N^2} \mathbf{w}^T \mathbf{X}^\mathbf{T} \mathbf{L} \mathbf{X} \mathbf{w}$$

$$\Rightarrow$$

$$\mathbf{w} = \left( \mathbf{X}_\ell^T \mathbf{X}_\ell + \lambda \ell \mathbf{I} + \frac{\mu \ell}{N^2} \mathbf{X}^T \mathbf{L} \mathbf{X} \right)^{-1} \mathbf{X}_\ell^T \mathbf{Y}$$

where

- $\mathbf{L}$ is the *Laplacian*, $\mathbf{L} = \mathbf{D} - \mathbf{S}$, where $\mathbf{S}$ contains the similarities, $\mathbf{D} = diag(sum(\mathbf{S}))$
- $\mathbf{X}_\ell$ is the matrix of labeled data (with samples in its rows)

$$
\begin{aligned}
\sum_{i,j} s_{ij}(f_i - f_j)^2 &= \sum_{ij} s_{ij}(f_i^2 + f_j^2 - 2f_i f_j) \\
&= \sum_{i,j} s_{ij} f_i^2 + \sum_{i,j} s_{ij} f_j^2 - 2\sum_{i,j} s_{ij} f_i f_j \\
&= 2\sum_{i,j} s_{ij} f_i^2 - 2\sum_{i,j} s_{ij} f_i f_j \\
&= 2\sum_{i,j} d_i f_i f_i - 2\sum_{i,j} s_{ij} f_i f_j \\
&= 2\mathbf{f}^T\mathbf{D}\mathbf{f} - 2\mathbf{f}^T\mathbf{S}\mathbf{f} = 2\mathbf{f}^T \underbrace{(\mathbf{D} - \mathbf{S})}_{\mathbf{L}} \mathbf{f} \\
&= 2\mathbf{w}^T\mathbf{X}^T\mathbf{L}\mathbf{X}\mathbf{w}
\end{aligned}
$$

# Optimized representation

- storing/representing data is often a challenge
- example: representing natural language documents in numeric format, using **bag-of-words** (BoW) representation
  - problem: can easily reach $10^5$ dimensions (= size of vocabulary)
  - question: how to store the data?
  - answer: in sparse format


BoW representation (from https://www.freecodecamp.org/news/text-classification-and-prediction-using-bag-of-words-8aeb1396cded/)

# Python: NumPy, SciPy, scikit-learn

- **NumPy**[5] – representation (vectors, matrices, sparse matrices, etc.) + basic operations (e.g. matrix multiplication, trigonometric functions, etc.)
- **SciPy**[6] – higher-level methods: eigenvalue problems, matrix decompositions, clustering, signal processing, etc.
- **scikit-learn**[7] – machine learning library (*traditional*, *GOF* models)

[5] https://numpy.org/
[6] https://scipy.org/
[7] https://scikit-learn.org/

## 1. DOK (Dictionary Of Keys)

```
from scipy.sparse import csr_matrix, dok_matrix, lil_matrix, random

A = [[0,0,1],
     [2,0,4],
     [3,0,0.]]

M = dok_matrix(A)
print(M.keys())
print(M.values())
```

Output:

```
dict_keys([(0, 2), (1, 0), (1, 2), (2, 0)])
dict_values([1.0, 2.0, 4.0, 3.0])
```

## 2. LIL (LIst of Lists)

```
A = [[0,0,1],
     [2,0,4],
     [3,0,0.]]

M = lil_matrix(A)
print(M.data)
print(M.rows)
```

Output:

```
[list([1.0]) list([2.0, 4.0]) list([3.0])]
[list([2]) list([0, 2]) list([0])]
```

## 3. CSR (Compressed Sparse Row)

```
A = [[0,0,1],
     [2,0,4],
     [3,0,0.]]

M = csr_matrix(A)
print(M.data)
print(M.indices)
print(M.indptr)
```

Output:

```
[1. 2. 4. 3.]
[2 0 2 0]
[0 1 3 4]
```

# EXERCISE 1.

Use the *Latest Netflix TV shows and movies*[8] dataset from Kaggle and find the most important/influential/… actors – using PageRank.

- Use the `cast` field/column of the dataset.
- Filter out the non-string records.
- Tokenize and lowercase the actor names.
- Use a `csr_matrix` or `csc_matrix` to build the adjacency matrix: $P(i \to j) = P(j \to i) \propto$ no. of common movies of $i$ and $j$.
- Run 10 iterations of PageRank.
- List the 10 most important actors.

[8] https://www.kaggle.com/datasets/senapatirajesh/netflix-tv-shows-and-movies

```
[('anupam kher', 0.0004560616609445015),
 ('shah rukh khan', 0.0003622616850521827),
 ('akshay kumar', 0.00030768864357521474),
 ('om puri', 0.0003005761049536558),
 ('naseeruddin shah', 0.00028859258037161 05),
 ('boman irani', 0.0002884407892481599),
 ('paresh rawal', 0.00028678620662588196),
 ('amitabh bachchan', 0.00027892696637877 55),
 ('takahiro sakurai', 0.0002602693953920269),
 ('yuki kaji', 0.00024605644926779867)]
```

# EXERCISE 2.

Use the *Tripadvisor Reviews 2023*[9] dataset and build a least squares model to predict the ranking assigned to a review.

- Filter out non-string reviews.
- Use `sklearn.feature_extraction.text.CountVectorizer` to build the document-term matrix.
- Extend $\mathbf{X}$ by a constant dimension of $1$.
- Split the dataset into training and test sets using `sklearn.model_selection.train_test_split` (80%–20%).
- Determine $\mathbf{w}$ using gradient descent ($\eta = 10^{-2}$, #iterations=$1000$).
- Calculate MSE.

[9] https://www.kaggle.com/datasets/arnabchaki/tripadvisor-reviews-2023

```
Iteration 0: delta=119.96651299468701
Iteration 100: delta=0.0464057804017199
Iteration 200: delta=0.023476858434091974
Iteration 300: delta=0.015798984143147184
Iteration 400: delta=0.012445411372045998
Iteration 500: delta=0.010512319843923675
Iteration 600: delta=0.009203779762576428
Iteration 700: delta=0.00823492218289109
Iteration 800: delta=0.007475758296139431
Iteration 900: delta=0.006857966694612957
---
MSE=0.023151239769759233
```