高速省メモリ線形代数

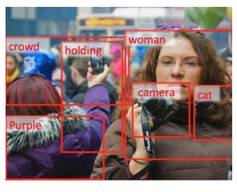
国立情報学研究所　特任研究員　松井勇佑

課題名 圧縮線形代数：
データ圧縮による省メモリ
高速大規模行列演算

解決したい問題

大量データ → 処理（線形代数） → 応用

$$\begin{bmatrix}2.42\\4.09\\1.31\\2.51\end{bmatrix}\begin{bmatrix}4.37\\0.11\\7.42\\3.21\end{bmatrix}\cdots \Rightarrow \begin{bmatrix}0.16&0.75&0.19&0.53\\0.10&0.37&0.21&0.91\\0.72&0.76&0.47&0.91\\0.17&0.89&0.13&0.51\end{bmatrix}\begin{bmatrix}0.22\\1.34\\0.19\\1.25\end{bmatrix}=\begin{bmatrix}2.42\\1.31\\5.08\\2.51\end{bmatrix}$$



データ量が多すぎるので

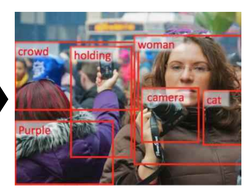
☹大量メモリ消費．遅い．

➢現状，大規模処理は力技（大量サーバ）
➢Google等の大企業しか取り組めない

提案：高速省メモリ演算体系

大量データ → コード → 処理（圧縮線形代数） → 応用

$$\begin{bmatrix}2.42\\4.09\\1.31\\2.51\end{bmatrix}\begin{bmatrix}4.37\\0.11\\7.42\\3.21\end{bmatrix}\cdots \xrightarrow{\text{圧縮}} \begin{bmatrix}32\\61\end{bmatrix}\begin{bmatrix}2\\93\end{bmatrix}\cdots \Rightarrow \text{圧縮線形代数}\ \begin{bmatrix}71&13\end{bmatrix}\begin{bmatrix}32\\61\end{bmatrix}=\begin{bmatrix}2\\93\end{bmatrix}$$



(1) 圧縮して　(2) 線形代数演算

例：10^9個の128Dベクトル（512 GB）を 8 GBに圧縮
➡ 100倍高速に線形代数演算

ACT-I期間の成果

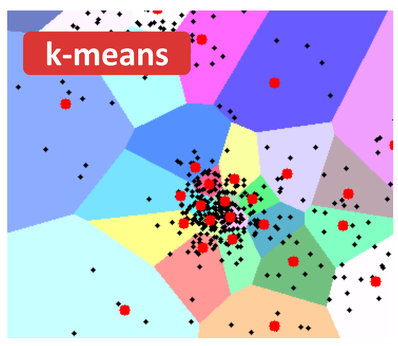
(1) 高速省メモリクラスタリングの実現
- Billion-scaleで世界最速レベル
- マルチメディア分野最高の国際会議 ACM Multimediaに採録
- Githubでコード公開．pipでインストール可

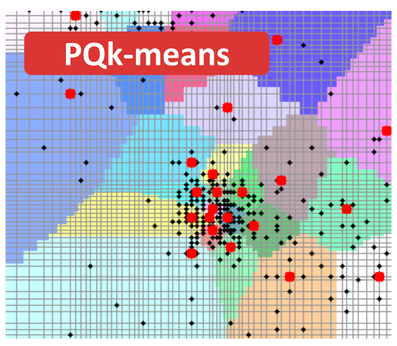
(2) パラメータ調整が必要ない近似最近傍探索手法の実現
- Billion-scaleで低精度だが世界最速
- マルチメディア分野最高の論文誌 IEEE Transactions on Multimediaに採録

サマリー

- PQk-means: 高速省メモリで billion 級のクラスタリング
- k-meansに比べ,
  - ✓ 10x から 100x 高速
  - ✓ 10x から 100x メモリ効率良

アイディア:
- 入力ベクトルを Product-quantized (PQ) でPQ-codeに圧縮
- PQ-domainでクラスタリング





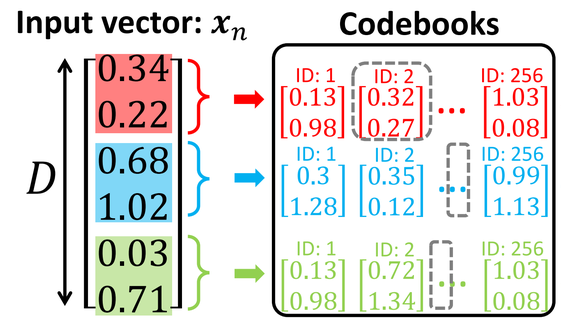
手法

Preliminary:
$\bar{\boldsymbol{x}}_n \leftarrow \mathrm{PQ}(\boldsymbol{x}_n)$ for each n

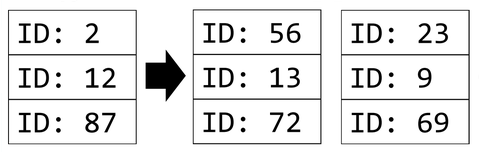
Main algorithm:
Input: $\bar{X} = \{\bar{\boldsymbol{x}}_1, \ldots, \bar{\boldsymbol{x}}_N\}$, K
Output: $\bar{M} = \{\bar{\boldsymbol{\mu}}_1, \ldots, \bar{\boldsymbol{\mu}}_K\}$

```
while(!convergence)
  a[] ← ∅
  for n = 1 to N
    k ← FindNN(x̄_n, M̄)
    a[k].Push(n)
  for k = 1 to K
    μ̄_k ← Update Center(X̄, a[k])
```

Product quantization [Jégou+, TPAMI 11]

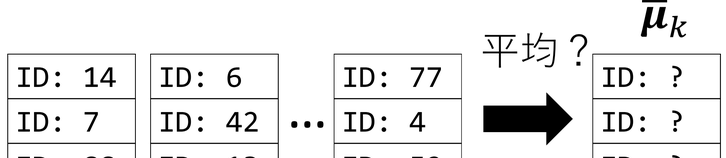


$\bar{\boldsymbol{x}}_n$ 近傍？ $\bar{M}$



1. Linear scan
2. PQTable [Matsui+, ICCV 15]

平均？ $\bar{\boldsymbol{\mu}}_k$



Sparse voting を提案
- 高速. 計算量: $O(M(N + KL\|\boldsymbol{h}\|_0))$

評価

ILSVRC2012 data (subset of ImageNet) を使って既存手法と比較
- 4096D AlexNet feature with 32-bit codes

Method	K	Error	Time [s]	Memory
PQk-means	10^2	65.09	18.2	5.12 MB
(proposed)	10^3	60.92	1.51×10^2	5.12 MB
Bk-means	10^2	66.35	12.3	5.12 MB
[Gong+, CVPR 15]	10^3	63.19	1.00×10^2	5.12 MB
k-means	10^2	64.25	9.12×10^3	21.0 GB
	10^3	58.95	1.06×10^5	21.0 GB
Ak-means	10^2	64.29	3.00×10^3	21.0 GB
[Philbin+, CVPR 07]	10^3	59.76	2.96×10^3	21.0 GB

Bk-meansのほうが少し早いが，精度は低い

k-meansとAk-meansは精度高いが，遅くメモリ消費大

Large-scale evaluation
- PQk-means
- $D = 96, B = 32$
- 全部で 4GB のみ
- 5 iterations

Dataset	N	K	Time
Deep1B	10^9	10^2	33 mins
		10^3	67 mins
		10^4	10 hours
		10^5	12 hours

billion級データをわずか一時間

コード

https://github.com/DwangoMediaVillage/pqkmeans

```
$ pip install pqkmeans
```

```
import pqkmeans
import numpy as np

Xt = np.random.random((1000, 128))
X = np.random.random((100000, 128))

# Train
encoder = pqkmeans.encoder.PQEncoder()
encoder.fit(Xt)

# Convert
X_pqcode = encoder.transform(X)

# Run clustering
labels =
pqkmeans.clustering.PQKMeans(encoder=
encoder, k=5).fit_predict(X_pqcode)
```