Designs of Algorithms and Programming for Massive Data

DS8001

Andriy Miranskyy

Nov. 21, 2016

Outline

- Parallelization Case studies
  - Gaussian Naïve Bayes (training)
  - k-means
- k-means++

Gaussian Naïve Bayes

What type of ML problems can we solve with this algorithm?

Assumptions

- The features are continuous variables
- The features are not correlated
- Each feature is normally distributed
- No missing or null values

Input data

- A table of the form

Row #	Feature 1	Feature 2	…	Feature N	Class
1					
2					
…					
M					

Testing

- Posterior = Prior × Likelihood ÷ Evidence
  - We can ignore the Evidence, as it is a normalizing constant

$$p(C_k|x_1, x_2, \ldots, x_n) = p(C_k) \times p(x_1|C_k) \times p(x_2|C_k) \times \cdots \times p(x_n|C_k)$$

- We need to precompute prior and likelihood probabilities in training
  - Prior: compute as a fraction of the number of C_k observations in the total number of observations
    - This may be biased, but we can ignore this from the algorithmic perspective
  - Likelihood: compute empirical mean and variance for each $p(x_i|C_k)$
    - Given normality assumption, this is enough to to estimate the distribution

Training – Sequential

```
#compute
For every class value $c$:
  $d$ <- a subset of data where class == $c$
  Compute prior probability for $c$
  For every feature $f$:
    Compute mean and variance for a given $f$ in $d$
```

How do we parallelize the training?

- What information do you know to make a decision?
- What are our goals?

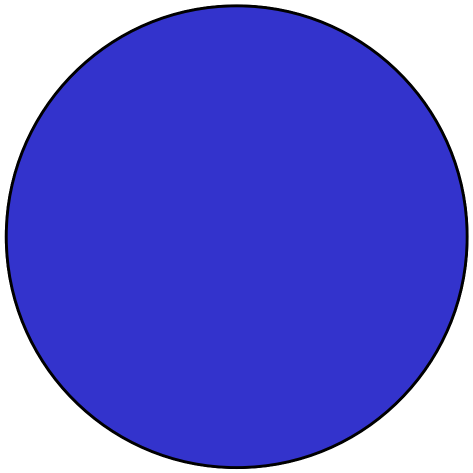
Questions and Goals

- What are the values of
  - Number of features *n*?
  - Number of rows *m*?
  - Number of classes *k*?
  - Number of processors *p*?
  - Number of compute nodes *s*?
- Goals
  - Maximize data locality
  - Minimize volume of data exchange
  - Minimize frequency of interactions

Group Work

- Split into groups (2-6 people)
- Discuss for 5 minutes:
  - How do we parallelize this algorithm?
  - What information do you know to make a decision?
  - What are our goals?
- Pick a reporter in your group and report your findings

5 minutes



Options

- Split by
  - Number of features *n*
  - Number of classes *k*?

- How does
  - *n*
  - *k*
  - $n \times k$
- relate to
  - *p* (number of processors)?

Training – Parallel

```
#compute
For every class value c do parallel:
  d <- a subset of data where class == c
  Compute prior probability for c
  For every feature f do parallel:
    Compute mean and variance for a given f in d
```

Options

- And what if the number of rows *m* is huge?
- Then we can parallelize computations even further
- By partitioning the dataset into a number of subsets and computing mean and variance independently and then performing the aggregation
    - One may have to compute mean and variance incrementally to avoid round-off error

- Is there an issue with this approach?

Training – Parallel – Data Split v.1

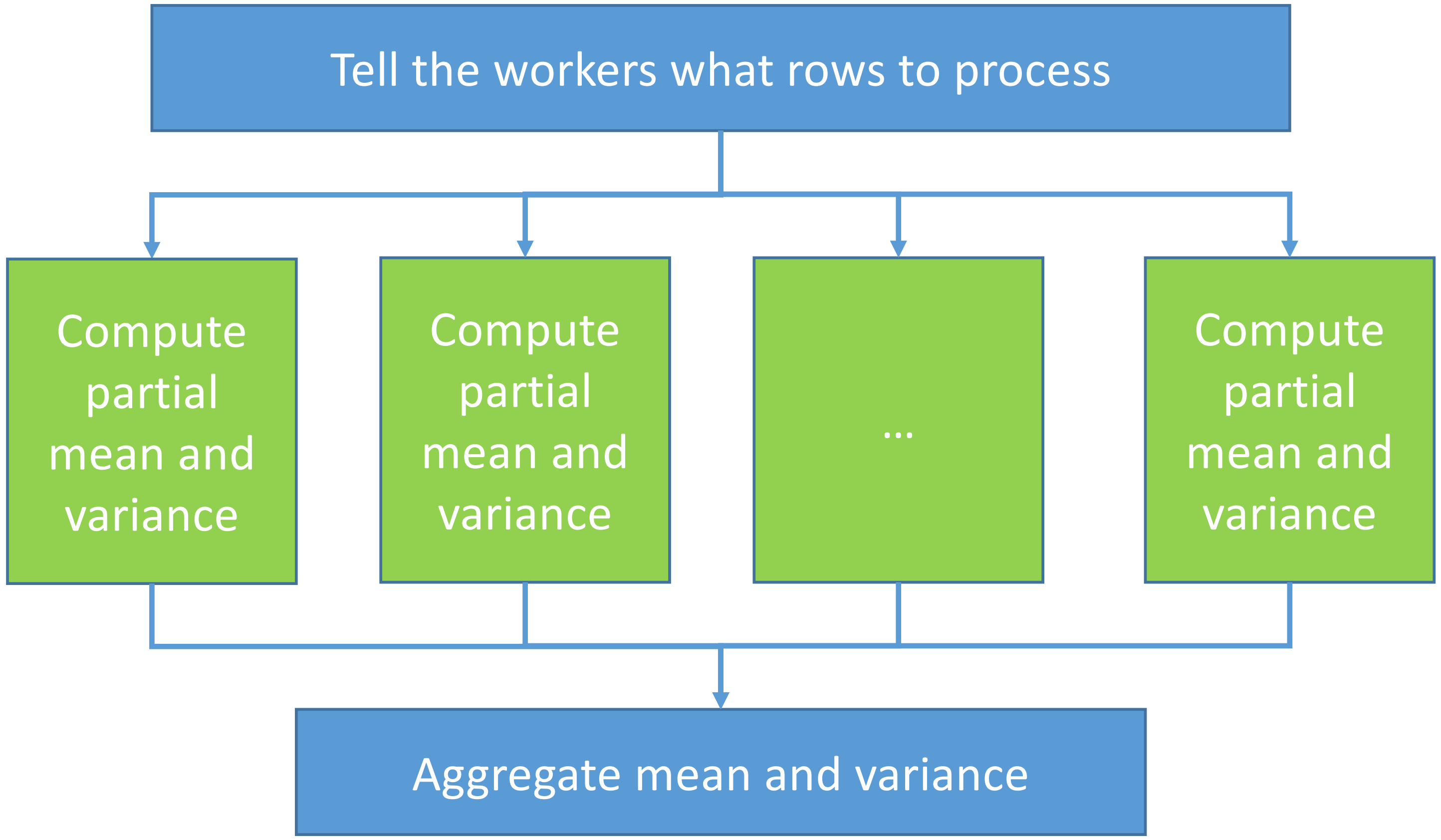
```
#compute
For every class value c do parallel:
  d <- a subset of data where class == c
  Compute prior probability for c
  For every feature f do parallel:
    For every data partition z do parallel:
      Partially compute mean and variance for a
         given f in z
    Aggregate and compute mean and variance for a
         given f in d
```

Filtering the data will take long time! Especially if it resides on different compute nodes, as we will have to move the data across the nodes, as the observations for a given class are probably scattered throughout the file. Solution?

Solution

- Work with the data available on a given compute node
- Then aggregate on a master node
- Similar to classic MapReduce example of counting words

Parallel


Tell the workers what rows to process
Compute partial mean and variance
Compute partial mean and variance
…
Compute partial mean and variance
Aggregate mean and variance

Training – Parallel – Data Split v.2

```
On the master (a.k.a. leader / primary) processor:
  estimate the number of observations available in the input file
  give each slave (a.k.a. follower / replica) processor its subset of rows

On every slave (a.k.a. follower / replica) processor:
  create hash map of maps with the following structure
     z[class]->[feature] = [incremental_mean[1], incremental_var[2]]
  for every line:
    c <- class of a given observation
    for every feature f:
      if not exists z[c]->[f]:
        z[c]->[f] = 0
      update incremental mean and variance for a given z[c]->[f]
    return z to master processor

On the master (a.k.a. leader / primary) processor:
  merge hash maps from all the processors and compute mean, variance, etc.
```

1. http://math.stackexchange.com/questions/106700/incremental-averageing
2. https://en.wikipedia.org/wiki/Algorithms_for_calculating_variance#Parallel_algorithm

Characteristics of the algorithm

- Static tasks
- Static and Regular interactions
- Task size: uniform
  - Provided that the file is properly partitioned
- Interaction type: read / write
  - Write at the beginning
  - Read at the end
- Two-way interaction
- Mapping based on data partitioning

Multiple compute nodes

- Will work fine even if the number of compute nodes $s > 1$
- Information is passed only at the beginning and end of the compute cycle
- Topology is known in advance
    - Final aggregation is performed on the master/leader node
        - In: files are already on the distributed file system
        - Out: resulting mean and variance objects (small)

Intermezzo

Time Latency

Grace Hopper on a nanosecond

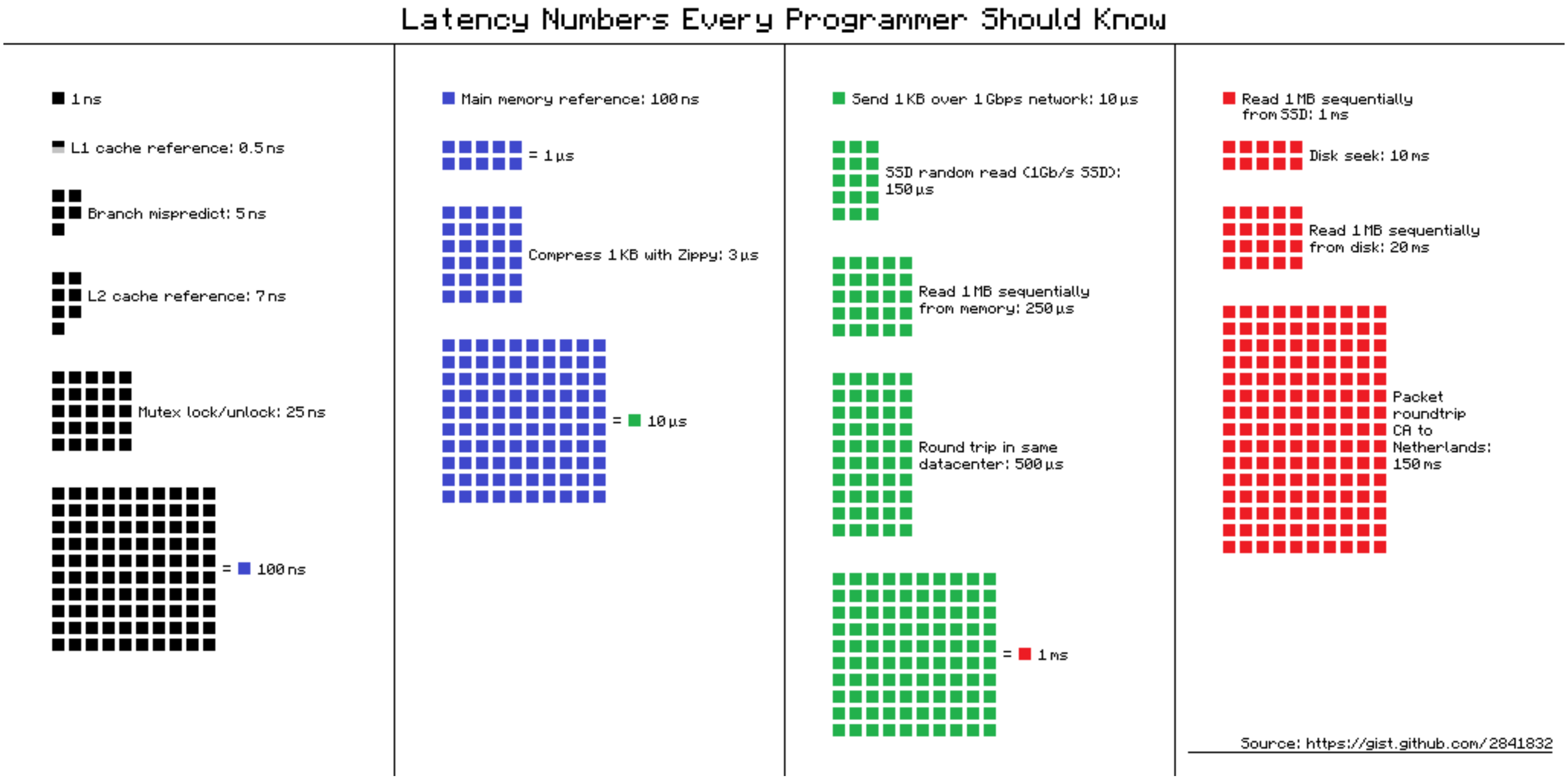
- https://www.youtube.com/watch?v=JEpsKnWZrJ8

Access time as of 2009

execute typical instruction	1/1,000,000,000 sec = 1 nanosec
fetch from L1 cache memory	0.5 nanosec
branch misprediction	5 nanosec
fetch from L2 cache memory	7 nanosec
Mutex lock/unlock	25 nanosec
fetch from main memory	100 nanosec
send 2K bytes over 1Gbps network	20,000 nanosec
read 1MB sequentially from memory	250,000 nanosec
fetch from new disk location (seek)	8,000,000 nanosec
read 1MB sequentially from disk	20,000,000 nanosec
send packet US to Europe and back	150 milliseconds = 150,000,000 nanosec

Taken from http://norvig.com/21-days.html#answers

Access time as of 2009



k-means

Clustering

- "Given an integer *k* and a set of *n* data points in R^d, the goal is to choose *k* centers so as to minimize ϕ, the total squared distance between each point and its closest center."[1]

- NP-hard problem
    - Can't compute in polynomial time
    - Can't verify in polynomial time

- Need a heuristic

1. D. Arthur and S. Vassilvitskii. "k-means++: The advantages of careful seeding." *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2007.

Sequential (Lloyd's algorithm[1,2])

```
Input: set of objects (points) O, number of clusters k

Randomly set centers of each cluster and save them in C
Assign each object randomly to one cluster
n = 1 // set number of changed objects

//while at least 1 object changes cluster
while( n > 0 ):
  n = 0
  for each o in O:
    nC = getNearestCluster(o, C)
    if( nC != o.nC ):
      n++
      o.nC = nC
  recalculate the center of each cluster
```

1. S. P. Lloyd. Least squares quantization in PCM. IEEE Transactions on Information Theory, 28(2):129–136, 1982. (Originally proposed in 1957)
2. Adopted from K. Stoffel and A. Belkoniene. 1999. Parallel k/h-Means Clustering for Large Data Sets. In *Proceedings of the 5th International Euro-Par Conference on Parallel Processing* (Euro-Par '99), 1451-1454.

Parallel

What should we pass from the workers nodes?

```
Input: set of objects O, number of clusters k

randomly set centers of each cluster and save them in C
assign each object randomly to one cluster
n = 1 // set number of changed objects

//while at least 1 object changes cluster
while( n > 0 ):
  n = 0
  for each o in O do parallel:
    nC = getNearestCluster(o, C)
    if( nC != o.nC ):
      n++
      o.nC = nC

recalculate the center of each cluster
```

Adopted from K. Stoffel and A. Belkoniene. 1999. Parallel k/h-Means Clustering for Large Data Sets. In *Proceedings of the 5th International Euro-Par Conference on Parallel Processing* (Euro-Par '99), 1451-1454.

Parallel

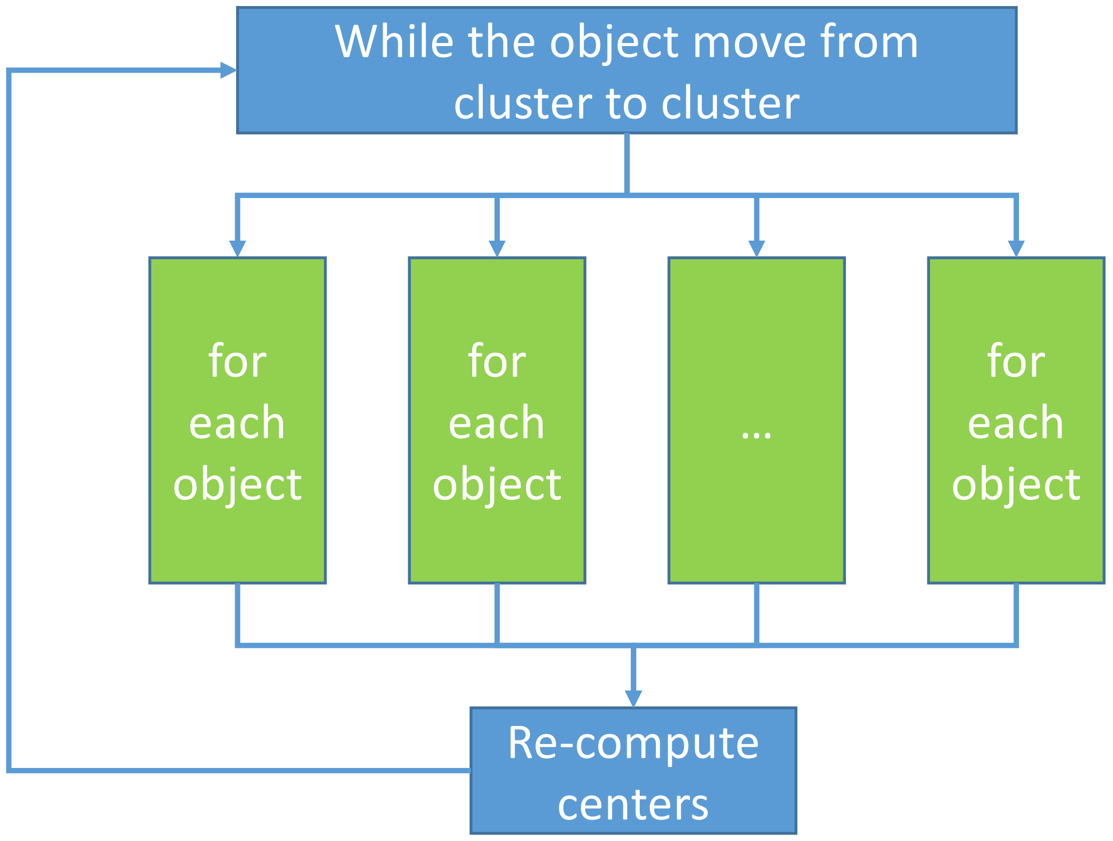
```
Input: set of objects O, number of clusters k

randomly set centers of each cluster and save them in C
assign each object randomly to one cluster
n = 1 // set number of changed objects

//while at least 1 object changes cluster
while( n > 0 ):
  n = 0
  for each o in O do parallel:
    nC = getNearestCluster(o, C)
    if( nC != o.nC ):
      n++
      o.nC = nC
    partially compute centers of clusters
recalculate the center of each cluster
```

Adopted from K. Stoffel and A. Belkoniene. 1999. Parallel k/h-Means Clustering for Large Data Sets. In *Proceedings of the 5th International Euro-Par Conference on Parallel Processing* (Euro-Par '99), 1451-1454.

Parallel


While the object move from cluster to cluster
for each object
for each object
…
for each object
Re-compute centers

Characteristics of the algorithm

- Static tasks
- Static and Regular interactions
- Task size: uniform
  - Provided that the file is properly partitioned
- Interaction type: read / write
  - Write at the beginning
  - Read at the end
- Two-way interaction
- Mapping based on data partitioning

Multiple compute nodes

- Information is passed iteratively
  - Need to minimize the amount of communication – pass around only partial means
- Topology is known in advance

k-means++

D. Arthur and S. Vassilvitskii. "k-means++: The advantages of careful seeding." *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2007.

Selection of centers

- We know that k-means is sensitive to the choice of the starting centers of clusters
- Can we improve the initial selection?

Initial selection of centers

1. Choose on cluster center at random from all the observations O
2. Choose a new center c_i with the probability of $D(x)^2 / \sum_{x \in O} D(x)^2$
    - $D(x)$ denotes the shortest distance from a data point to the closest center we have already chosen
3. Repeat step 2 until all *k* centers are chosen.

What does “with probability” means? Example

- O = [0, 2, 7, 8]
- Pick the first center at random, say 2
- Compute D(x):
  - 0: $(2-0)^2 = 4$
  - 7: $(2-7)^2 = 25$
  - 8: $(2-8)^2 = 36$
- Compute p-values:
  - 0: 4 / (4+25+36) = 0.06
  - 7: 25 / (4+25+36) = 0.38
  - 8: 36 / (4+25+36) = 0.55
- Compute cdf: 0.06, 0.45, 1.00
  - Note round-offs

```
u = rand()
if(u ≤ 0.06):
  return 0
else if(u ≤ 0.45):
  return 7
else:
  return 8
```

Accuracy

- The initial selection guarantees[1] that

$$E[\phi] \leq 8(\ln k + 2)\phi_{OPT}$$

- where

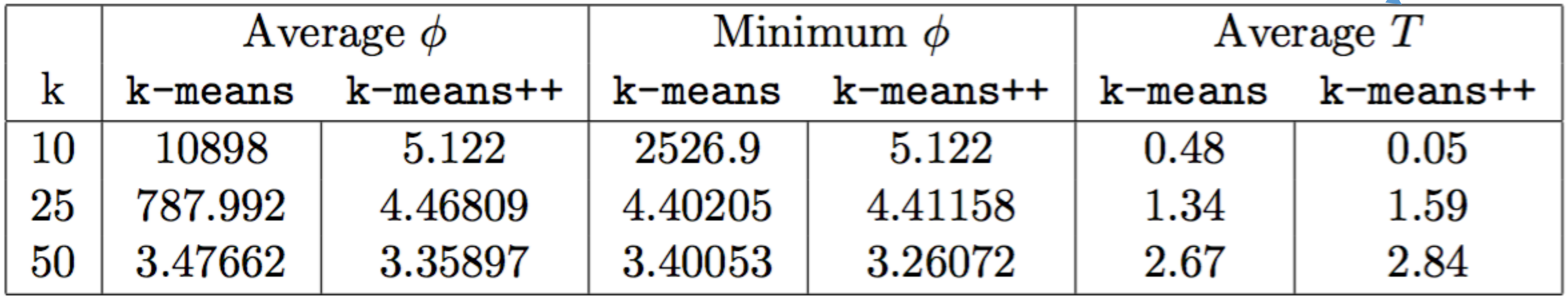
$$\phi = \sum_{x \in O} \min_{c \in C} \|x - c\|^2$$

- and ϕ_{OPT} is the optimal selection of objects

1. D. Arthur and S. Vassilvitskii. "k-means++: The advantages of careful seeding." *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2007.

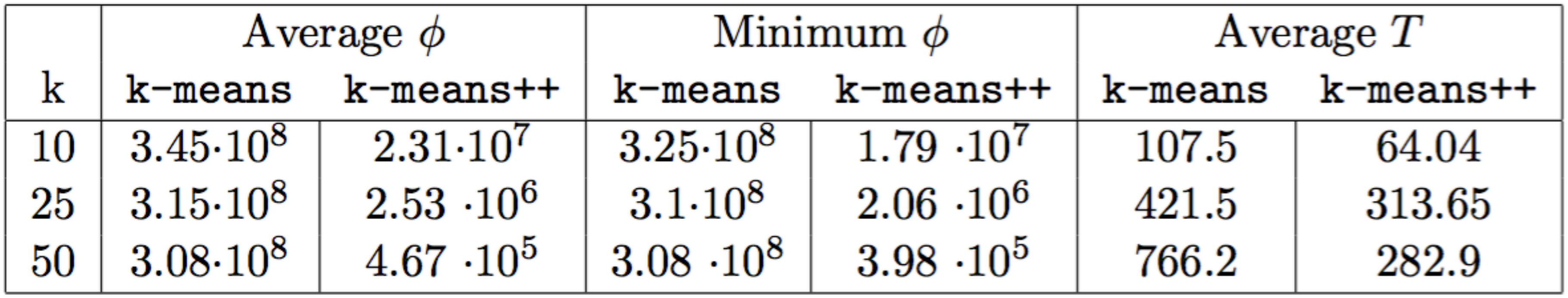
k-means vs. k-means++

T time needed to compute the output



k	Average ϕ		Minimum ϕ		Average T	
	k-means	k-means++	k-means	k-means++	k-means	k-means++
10	10898	5.122	2526.9	5.122	0.48	0.05
25	787.992	4.46809	4.40205	4.41158	1.34	1.59
50	3.47662	3.35897	3.40053	3.26072	2.67	2.84

Table 1: Experimental results on the *Norm-10* dataset (n = 10000, d = 5)



k	Average ϕ		Minimum ϕ		Average T	
	k-means	k-means++	k-means	k-means++	k-means	k-means++
10	$3.45 \cdot 10^8$	$2.31 \cdot 10^7$	$3.25 \cdot 10^8$	$1.79 \cdot 10^7$	107.5	64.04
25	$3.15 \cdot 10^8$	$2.53 \cdot 10^6$	$3.1 \cdot 10^8$	$2.06 \cdot 10^6$	421.5	313.65
50	$3.08 \cdot 10^8$	$4.67 \cdot 10^5$	$3.08 \cdot 10^8$	$3.98 \cdot 10^5$	766.2	282.9

Table 4: Experimental results on the *Intrusion* dataset (n = 494019, d = 35)

Tables taken from D. Arthur and S. Vassilvitskii, 2007

k-means vs. k-means++

- k-means++ is often
  - Faster than k-means
  - More accurate

Parallelization

- k-means++ can be parallelized using MapReduce paradigm[1,2]

1. B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii, Scalable k-means++. Proceedings of the VLDB Endowment, 5, 7, pp. 622-633, 2012.
2. Y. Xu, W. Qu, Z. Li, G. Min, K. Li and Z. Liu, "Efficient k-Means++ Approximation with MapReduce," in IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 12, pp. 3135-3144, 2014.

Summary

- Naïve Bayes
- k-means
- k-means++

Cheat sheets

Characteristics of Tasks

- Generation: Static or Dynamic
  - Matrix multiplication vs. 15-puzzle
- Size: Uniform or Non-Uniform
  - Tasks are of the same size or not
  - Matrix multiplication vs. LU-decomposition
- Size of Data Associated with Tasks: Small or Large

Characteristics of Task Interactions

- Regular Interactions vs. Irregular Interactions
  - Topology: Know in advance vs. Unknown
  - Image dithering vs. Space matrix multiplication

- Read only vs. Read-Write

- One-way or Two-way

Mapping techniques

- Mappings must minimize overheads
- Primary overheads are communication and idling
- Minimizing these overheads often represents contradicting objectives

Schemes for Static Mapping

- Mappings based on data partitioning
- Mappings based on task graph partitioning
- Hybrid mappings