Large Scale Machine Learning 2018-03-27

Practical 2: Systems for Large Scale Machine Learning

---

1 Bonferroni's Principle

Question 1
We believe that, periodically, bandits gather at the same hotel to plot attacks. We wish to identify bandits, and will therefore consider as suspicious any two people who are both staying at the same hotel on two different nights. Let us make the following assumptions:

- We consider 1 billion people.
- Everyone of them goes to a hotel 1 night every 100 nights.
- There are 100 000 hotels, each holding 100 people every night.
- We have records for a period of 1 000 days.

Now imagine there are no bandits among the people we consider: everyone actually behaves at random, deciding for each night whether to go to a hotel (with probability 1% at random) and, if yes, choosing at random to which hotel among the 100 000 to go.

(a) How many suspicious events (= 2 people staying at the same hotel twice in the time period considered) will we find?

Solution: Let's call A the event "2 people choose, on the same night, to stay at a hotel", B the event "2 people who have chosen to stay at a hotel that night pick the same hotel", and C a suspicious event. Then

$$\mathbb{P}(A) = 10^{-2} \times 10^{-2} = 10^{-5}$$
$$\mathbb{P}(B) = \mathbb{P}(A) \times 10^{-5} = 10^{-9}$$
$$\mathbb{P}(C) = \mathbb{P}(B) \times \mathbb{P}(B) = 10^{-18}$$

The number of events to consider is the number of pairs of people times the number of pairs of days, which is approximately $\frac{1}{2}(10^9)^2 \times \frac{1}{2}(10^3)^2 = 25.10^{22}$.
Hence the number of a suspicious events is $10^{-18} \times 25.10^{22} = 250\,000$.

(b) How does this number change if we have records for 2 000 days?

Solution: The number of pairs of days becomes $\frac{1}{2}(2.10^3)^2$ and hence the number of suspicious events is multiplied by 4.

(c) How does this number change if we consider 2 billion people, and 200 000 hotels?

Solution: The probability of an event being suspicious is 4 times smaller (because $\mathbb{P}(B)$ is halved). The number of pairs of peoples to consider is 4 times larger. The number does not change.

(d) How does this number change if we consider as suspicious any two people who are aboth at the same hotel on *three* different days?

Solution: The probability of an event being suspicious is now multiplied by 10^{-9}, hence so is the number of suspicious events, which becomes smaller than 1. Now a suspicious event is really suspicious...

2 Similarity join with MapReduce

We will consider a problem known as the *similarity join*. We are given a large set $\mathcal{S}$ of elements and a similarity measure $s : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ that tells us how similar two elements of $\mathcal{S}$ are. We assume that s is symmetric: $s(x, y) = s(y, x)$. The goal of the similarity join is to return all pairs of $\mathcal{S} \times \mathcal{S}$ that have a similarity greater than a threshold $t \in \mathbb{R}$.

For example, $\mathcal{S}$ could be a set of images and s a function cleverly designed to be such that two images representing the same object have a large similarity. The goal of the similarity join would be to find all pairs of images that represent the same object. (In practice, definining such a function s is very hard problem.)

Question 2 Naive MapReduce

Consider a MapReduce implementation of the similarity join in which each Map task produces a key-value pair $\{\{i, j\}, (x_i, x_j)\}$, where i and j are indices of elements of $\mathcal{S}$ and x_i and x_j are the elements themselves. Note that $\{i, j\}$ is an unordered pair, meaning that $\{i, j\} = \{j, i\}$.

(a) What would the corresponding Reduce function be?

Solution: Apply s, compare the output to t, and output only if the value of s is greater than t.

(b) What is the reducer size of this algorithm?

> **Solution:** The reducer size is the upper bound on the number of values that are allowed to appear in the list associated with a single key. There are only two elements associated with any given key, so that size is $q = 2$.

(c) What is the replication rate of this algorithm?

> **Solution:** The replication rate is the number of key-value pairs per input. For each element x, one produces as many key-value pairs as there are elements $y \neq x$ in $\mathcal{S}$, that is, $r = |\mathcal{S}| - 1$.

(d) Suppose $\mathcal{S}$ contains 10 million images (a thousandth of the size of Flickr in 2015[1]), each of size 3.25 MB (the average size of a Flickr photo in 2016[2]). What is the total number of bytes communicated from Map tasks to Reduce tasks?

> **Solution:**
>
> — Number of Reduce tasks = number of unordered pairs of elements = $1/2 \times 10^7 \times (10^7 - 1) \approx 0.5 \times 10^{14}$.
>
> — Size of the input of a Reduce task, in bytes = size of two images = $2 \times 3.25 \times 10^6$.
>
> Hence: the total number of bytes communicated from Map tasks to Reduce tasks is about 3.25×10^{20}.

(e) How long would transfering data from a Map task to a Reduce task take, considering cluster nodes connected by Cat7 Ethernet cables, which operate at speeds of 10 Gbit/s? Imagining all data would have to go through the same switch (which is not usually the case), how long would communicating from all Map tasks to all Reduce tasks take?

Note: 10^8s ≈ 3 years and 1 B $= 8$ bit.

> **Solution:** 10 Gbit/s $\approx 1.25 \times 10^9$ B/s
>
> — From one Map task to one Reduce task: $2 \times 3.25 \times 10^6$ B; this is very little data, which fits well in memory and can be transmitted in about 5 ms. ($2 \times 3.25/1.25 \approx 5$.)
>
> — However, there are many Reduce tasks. Transmitting all the data would take $(5 \times 10^6) \times (0.5 \times 10^{14})/10^9 = 2.5 \times 10^{11}$ s, which is about 7 500 years.

[1]`https://blog.flickr.net/en/2015/05/07/flickr-unified-search/`

[2]`https://code.flickr.net/author/archieflickr/`

Question 3 Grouping pictures

Let us now group our elements into G groups $\mathcal{G}_1, \mathcal{G}_2, \dots, \mathcal{G}_G$ of $|\mathcal{S}|/G$ elements each. Consider a Map function that produces, for any input $\{i, x_i\}$, $(G-1)$ key-value pairs $\{(\{u, v\}, (i, u, x_i))\}_{v=1,\dots,G, v\neq u}$ where u is the index of the group that contains element i and v goes over the indices of all the other groups.

(a) What would be the corresponding Reduce function? Make sure there are no redundant operations.

Solution: Consider key (u, v). The corresponding value list contains all elements (j, g, x_j) belonging to either group $g = u$ or group $g = v$. The reducer must then compute $s(x_i, x_j)$ and compare it with t for all (i, j) where

- $i \in \mathcal{G}_u$ and $j \in \mathcal{G}_v$ (compare elements of different groups);
- $i \in \mathcal{G}_u$ (compare elements of the same group) if $v = u+1$. This last constraint imposes that only one of all the reducers with key $\{u, .\}$ compares elements of group $\mathcal{G}_u$.

(b) What is the reducer size of this algorithm?

Solution: The number of elements associated with any key $\{u, v\}$ is the number of elements belonging to either $\mathcal{G}_u$ or $\mathcal{G}_v$. Hence $q = 2\frac{|\mathcal{S}|}{G}$.

(c) What is the replication rate of this algorithm?

Solution: For each element x, one produces as many key-value pairs as there are groups different from the one x belongs to, that is, $r = G - 1$.

(d) Let us now consider 10 000 compute nodes and the same Flickr data as previously. How much data (in bytes) must fit in the main memory of compute nodes executing Reduce tasks?

Solution: The input to a Reduce task is now, in bytes, $(2 \times \frac{10^7}{10^4}) \times (3.25 \times 10^6) = 6.5 \times 10^9$. Hence the compute nodes executing Reduce tasks must have at least 6.5 GB of main memory.

(e) Still considering Cat7 Ethernet connections, how long would transfering data from a Map task to a Reduce task? Imagining all data would have to go through the same switch, how long would communicating from all Map tasks to all Reduce tasks take?

Solution: Each Reduce task receives 6.5 GB of data, which takes approximately 5 s to transfer on 10 Gbit/s Ethernet.

There are as many Reduce tasks as unordered pairs of groups, that is, $1/2 \times 10^4 \times (10^4 - 1) \approx 5 \times 10^7$. Our total communication time is now down to 2.5×10^8 s, which is about 7 years.

(f) What happens if we only take 100 groups?

Solution: Each group now contains 100 times more images, so the input to a Reduce task is 100 times larger: we need 650 GB of main memory. That is a lot, but you can have 1 TB of RAM on a single compute node.

Transfering the data to one Reduce task now also takes 100 times longer, that is, 500 s which is about 8 min.

The number of Reduce tasks, being the number of unordered pairs of groups, is now 10^4 times smaller. Our total communication time is reduced by a factor 100, bringing it down to about 1 month.

3 PageRank

Question 4 Computing PageRank

Consider the web graph on Figure 1.

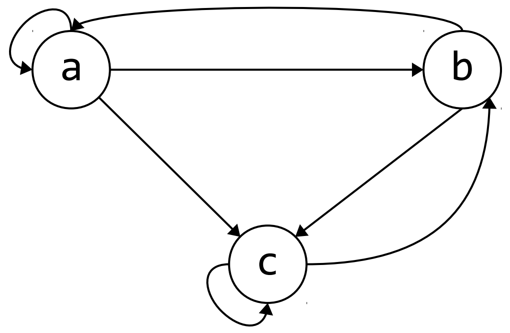

Figure 1: Example of a web graph.

(a) Why can you avoid considering taxation?

Solution: Taxation is used to deal with dead ends (pages with no link out) or spider traps (groups of pages with no link out.) Here a, b, and c all have links out so there are no dead ends. {a, b} has links to c, {a, c} has links to b and {b, c} has one link to a so there are no spider traps.

(b) Compute the PageRank of each page.

Solution: The transition matrix describes where a random surfer that starts from each node ends up after one step. For figure 1, it is

$$M = \begin{bmatrix} 1/3 & 1/2 & 0 \\ 1/3 & 0 & 1/2 \\ 1/3 & 1/2 & 1/2 \end{bmatrix}$$

There are no dead ends and the graph is strongly connected (you can get from any one node to any other node.) Therefore the distribution of the positions of a random surfer is given by the principal eigenvector of M.

Because the matrix is small, we can compute this principal eigenvector using Python:

```
import numpy as np
m = np.array([[1./3, 1./2, 0.],
              [1./3, 0, 1./2],
              [1./3, 1./2, 1./2]])
v, w = np.linalg.eig(m) # eigenvalue decomposition of m
w1 = w[:, v.argmax()]    # principal eigenvector
print w1/np.sum(w1)      # normalize
```

We get the following PageRank scores: PageRank(a) $\approx$ 0.231, PageRank(b) $\approx$ 0.308, PageRank(c) $\approx$ 0.462.

You can also solve the equation $\vec{v} = M\vec{v}$ by Gaussian elimination and get the exact values PageRank(a) = 3/13, PageRank(b) = 4/13 and PageRank(c) = 6/13.

In practice, with a very large graph, this is not feasible and we'd rather iterate applications of M to a vector $\vec{v_0} = [1/3, 1/3, 1/3]^\top$.

```
# raise m to the power of 10
np.linalg.matrix_power(m, 10)
# raise m to the power of 20
np.linalg.matrix_power(m, 20)
```

We see that the limit of $M^n\vec{v_0}$ gives us similar scores.

(c) What is the compact representation of the transition matrix of the web graph from figure 1?

Solution:

Source	Degree	Destination
a	3	a, b, c
b	2	a, c
c	2	a, b

(d) When does this compact representation take less space than the orginal one?

Solution: When the transition matrix is sparse, which is usually the case with web graphs.