Big Data Computing, Spring 2016/17
Homework 1

Marco Favorito
Master of Science in Engineering in Computer Science
Department of Computer, Control, and Management Engineering
University of Rome "La Sapienza"
favorito.1609890@studenti.uniroma1.it

June 11, 2017

Contents

1 Problem 1

We will use the following notation:

- m: number of element of the stream (only a working hypothesis, the approach still holds for unbounded stream);
- $N = \{1, 2, \ldots, n\}$: set of members (the domain);
- m_i: number of occurrences of the item $i \in N$;

Algorithm

Here we show the procedure:

- initialize k counters c_i to zero and k memory location M_i to ∞;
- upon the element a_i of the stream arrives, for each k:
  - compute the min-hash of a_i, i.e. $h_k(a_i)$;
  - if the value is lower than the one already stored (let's say M_i), then store it in the memory location and set the counter to zero; else if $M_k = h_k(a_i)$ then increment the counter. Otherwise ignore that element.

The estimate X_i of a single counter for the second moment F_2 is simply nc_i^2. The final estimate can be obtained as the following:

- group each X_i in s_1 small groups of s_2 estimates such that $s_1 s_2 = k$.
- take the average Y_j of each subgroup from $X_{j,1}, \ldots, X_{j,s_2}$ (i.i.d., by construction), then take the median from $Y_1, \ldots, Y_{s_1}$

In the next section we analyze how to reach an (ϵ, δ)-approximation.

Analysis

Theorem 1. *Let X be the estimate of F_2 described above. Then X is an unbiased estimator of F_2*

Proof. Since h_i for $i = 1 \ldots k$ are min-hash function, they satisfy the *min-wise independence* property, which means that each element of the domain N has equal probability to be the min-hash for that function. Moreover,

each counter c_i, at any given moment, is equal to the number of item a_i seen so far (i.e. m_i). Then it follows that:

$$E(X) = \frac{1}{n}\sum_{i=1}^{n} nc_i^2 = \sum m_i^2$$

□

Moreover, consider the variance of a single average Y_j.

$$E(X^2) = \frac{1}{n}\sum_{i=1}^{n}(nc_i^2)^2 = n\sum m_i^4 \leq nF_2^2$$

From this and from the assumption of i.i.d., we can say:

$$Var(Y_j) = \frac{Var(X)}{s_2} \leq \frac{E(X^2)}{s_2} = \frac{nF_2^2}{s_2}$$

Applying Chebyshev:

$$Pr[|Y_j - F_2| > \epsilon F_2] \leq \frac{Var(Y_j)}{\epsilon^2 F_2^2} \leq \frac{nF_2^2}{s_2\epsilon^2 F_2^2} \leq \frac{1}{8}$$

Where the last inequality follows from the choice of $s_2 = \frac{8n}{\epsilon^2}$. In the same fashion of [1], by the usual median trick [2][3], running in parallel $s_1 = 2\log(\frac{1}{\delta})$ instances, we reach the desired δ probability.

2 Problem 2

Thanks to the results from [4][9][8], we can use PCA (through SVD, for example) for perform a k-means over the data in order to find the two clusters.

The intuition is that, to some extent, PCA and k-means do the same thing: PCA does a projection of the data vectors in a suitable eigenspace, trying to minimize the mean-square error of the reconstruction; k-means tries to do the same thing, but finding some representatives (through mean-square error minimization) in the space of the data, and representing the data vectors with "indicator" vectors, with all zero except a 1 in the index which refers to the cluster to whom the vector has been assigned (aka membership indicators vectors).

The main difference, however, is that in order to minimize the objective function for k-means, thanks to the special formulation of the problem in [4], it turns out that if we want to use PCA, we need a *spectral relaxation* of k-means [9], where the membership indicator matrix is not constrained to have integer values in $\{0, 1\}$, but they can span in $[0, 1]$.

As fully described in [8], the procedure for this problem, called "PCA-guided search for K-means" is the following:

1. Perform standard K-means clustering in the PCA subspace;
2. Use the cluster membership obtained in Step 1 to construct initial cluster centroids in the full space. Perform standard K-means clustering in the full space.

In the following I'll try to summarize both the step and explaining them.

2.1 Step 1

As follows from Theorem 1 and 2 in [8], we know that the optimal solution for the relaxed (in continuous space) version of K-means lies in the k-dimensional PCA subspace. So, after this step, we have all the membership indicators vectors for every point of the data, computed directly from the projection of the same data in the first k component, plus an applied rotation.

2.2 Step 2

From the Step 1, we have the continuous membership indicators for every data point. From them, we retrieve the centroids of our k clusters (in our problem, just two), and set them as the centroids for our first iteration of K-means in the full space. Some important considerations:

- The reconstructed centroids are usually not local optimal solutions in full space, although they are local optimal solutions in PCA subspace.
- Usually, k-dimensional space is much smaller than the d-dimensional space. Due to this point, our approach achieve better performance in terms of time and quality:
  - Better time cost, since the most expensive operations of standard K-means are done for minimizing the objective function in the big d-dimensional space, with obvious drawbacks in terms of performances, which highly depends on the centroid initialization. In our approach, the centroids are found still executing standard K-means, but in a narrower space (since Theorem 2 guarantees us that the optimal solution lies in that subspace). The number of iterations in Step 2 is usually relatively small, because the initially defined cluster centroids are already close to a local minima.
  - Better solution quality, since it may happen that the standard K-means over the full-space might fall in a sub-optimal local minima more likely. Our approach, instead, since it searches in a narrower space, is more immune to noisy data, and it is more likely to find the global optimal solution.

3 Problem 3

3.1 Metric space

Here we propose a distance metric as the following:

- for each point of the space (i.e. the user), compute k components, which represent the number of seen movies for that category (which will be normalized on the number of movies). It is needed since the dimensionality is very high, and cluster the movie should be a good choice.[1]
- for the distance between two users we can use any3 distance we want (e.g. Minkowski, since we are dealing with low numbers) in an Euclidean space.

Each user is seen as a vector of k component, each of them representing the k-ity of that user. The distance between users is simply the sum of the differences between their k-ities (i.e. Minkowski).
Notice: one might be tempted to use Jaccard similarity for the similarity on the categories. However, if the number of movies is high, sparsity may heavily affect the Jaccard Index efficacy.

3.2 Read the data in one-pass

For each user keep k counters, that represent the number of movies seen for the category k. At the same time, keep a distinct element counter for the total number of seen film for each category. Assuming that the number of movies is not too large, we can use an exact counter, implemented with hash tables, then we can ignore this step in the analysis.

3.3 Complete algorithm and analysis

Algorithm 1 consists in a run of the CURE algorithm, fully described in [5]. Here we will provide a summary for estimate the time and space complexity.

Preprocessing: the preprocessing costs $O(n)$ time and space. It consists in simply counting and read from disk one edge at a time.

[1] As explained in [6], Cap. 9.3.3, Jaccard Index or Cosine Similarity perform bad in high dimensionality. Moreover, in future applications, we can cluster *items* and users in an iterative way.

Algorithm 1 Cluster users by seen movies

```
1: function CLUSTER(G)
Require: G = (V, E), bipartite graph user-movie
2:     Û ← 0_{n×k} #Array of processed users
3:     C ← 0_{k×1} #Distinct element counters for each category
4:     for edge (i, j) ∈ E do
5:         Û[i][category(j)] + +
6:         C[category(j)].insert(j)
7:     end for
8: # CURE, as described in [5], Section 4
9:     Draw random sample S from Û
10:    Partition sample
11:    Partially cluster partitions
12:    Cluster partial clusters
13:    Label every u ∈ Û
14: end function
```

Draw random sample: given that n is the number of users, we should find our k clusters[2] on a smaller sample. Theorem 4.1 in [5] explain which is the ideal size of our sample S, giving some guarantee, with a tunable probability δ, about the size of the clusters in the sample.
It has no cost in space: all operations are done in memory.

Partition sample The idea is that we can reduce the running time by execute more cluster task in parallel, one for each partition. Then we can choose the number of partition, p, and a parameter q, such that:

- the improvement factor is maximized, i.e. $\frac{q-1}{pq} + \frac{1}{q^2}$
- $\frac{n}{pq}$ (i.e. the number of computed clusters in each partition) is no greater than 2 or 3 times the desired k.

Partially cluster partitions AND Cluster partial clusters The hierarchical clustering algorithm proposed by CURE can be summarized in the following:

- Take as input the set of points to cluster S and the desired number of clusters k;

[2] Be careful: this k is not the same of the number of categories!!!

- Start with each point as a cluster, with the attributes .closest (the closest cluster), .mean (its centroid), .rep (set of c representatives, c chosen a priori).
- Build a heap Q with increasing .closest distance. Build a k-d tree T (which is used to efficiently find the closest cluster for every new merged cluster, $log(n)$).
- While the number of cluster is not equal to k, merge the closest pair of cluster (taken from the heap Q).
- the most expensive operation is the merge of two clusters: every time, we need to find the *well-scattered* c points that will be the new representatives (the overall cost is n^2). In this operation, we compute the new .mean and .rep; moreover, we shrink each representative by a factor α toward the mean, which gets rid of surface abnormalities and mitigates the effects of outliers [5].

The overall cost of this operation is $O(n^2 log(n))$, which can be improved, as we said before, partitioning the sample appropriately.

Labeling After we found the clusters, all we need is to assign each sample of the dataset to the cluster of the closest representative. This, as said in [5], enables CURE to, in the final phase, correctly distribute the data points when clusters are non-spherical or non-uniform. It costs another pass over the data, obviously.

3.4 Find the ideal k

In order to find the best k that fits the data, we can use the strategy described in [6], Section 7.3.3: do a binary search over the k, considering the resulting average diameter of the clusters: when the changes are little, change search direction.

4 Problem 4

In the following, I'm assuming that G is a *directed* graph.
The solution for this problem employs the reservoir algorithm [7] initializing a sample of size k for each node, as an adjacency list, and n counters for the number of seen outgoing edges from a node i, for every node.
When the sample size become greater than k, then apply the rule of the algorithm, i.e.:

- Assume n is the number of edges (i.e. nodes in our graph representation) seen so far. With probability $\frac{k}{n+1}$ pick the next edge you see.
- if the $n+1_{th}$ edge is not picked, keep the sample of k element as it is.
- otherwise, select uniformly at random one element of the sample and replace it with the new entry.

4.1 Pseudocode

Algorithm 2 Graph sampling

```
1: function SAMPLING(G)
Require: G = (V, E), directed graph with n = |V|
2:     Ĝ ← ⊥_{n×k} #Adjacency list
3:     C ← 0_{n×1} #Counters for each node of seen edges
4:     for edge (i, j) ∈ E do
5:         if C_i < k then
6:             G[i][C_i] ← j
7:         else with probability k/(C_i+1):
8:             pick j
9:             select uniformly at random r = 0, ..., k − 1
10:            G[i][r] ← j
11:        end if
12:        C_i ← C_i + 1
13:    end for
14: end function
```

4.2 Analysis

The algorithm passes only once over the data. Its space is $o(n(k+1)\log n) \approx o(nk)$, as required.

In the following we will focus on the most important properties that should be preserved by a sample of a graph, and analyzing our performances. Obviously, graph properties depends on the kind of graph we are dealing with. We will deal mainly with two well-known network models: the random graph $G(n,p)$[3] and the scale-free network model[4].
For the random graph $G(n,p)$, we assume p the probability of picking an edge between two nodes (taking into account the order).
For the scale-free network, we say that the degree distribution[5] follows a power law, i.e.:

$$P(k) \sim k^{-\gamma} \tag{1}$$

where $P(k)$ is the fraction of nodes with degree k and γ is a parameter whose value is typically in the range $2 < \gamma < 3$.

4.2.1 Number of nodes/edges

The main advantage is that we store every vertex of the original graph, i.e. no loss over the domain of individuals.
The number of edges is underestimated, since we drop edges of "big" nodes (i.e. more than k outgoing edges).
Random graph:

The expected number of edges is: $E(m) = n(n-1)p$

The expected average degree c is: $E(c) = E(\frac{m}{n}) = \frac{n(n-1)p}{n} = (n-1)p$

If $E(c) > k$, we can write: $(n-1)p = k + l$, where l is the expected number (proportion, let's say) of dropped edges per node.

The overall expected number of dropped edges in our sample is: $nl = n[(n-1)p - k]$
Scale-free network: The expected number of edges is:

$$E(m) = \sum_{d=1}^{\infty} nd\frac{1}{d^{\gamma}} = n\sum_{d=1}^{\infty}\frac{1}{d^{\gamma-1}}$$

Considering the range of γ above defined, The sum is upper-bounded by the harmonic number H_{∞} ($\gamma = 2$) and lower-bounded by $\frac{\pi^2}{6}$ ($\gamma = 3$). [6]

[3] aka Erdős–Rényi (ER) model.

[4] Very famous examples of this kind of network are: social networks/collaborative networks, computer networks, financial networks, Protein-protein interaction networks, semantic networks.

[5] Since we are talking about directed graph, we should define two degree distribution, one for the in-degree and the other for the out-degree values. When needed, in the following we will distinguish this two kind of distribution.

[6] Thanks, Euler.

Thanks to our approach, in order to compute the expected number of edges of our sample, we must stop the series to $d = k$, and then keep summing $\frac{1}{k^{\gamma-1}}$. In symbols:
Upper bound ($\gamma = 2$)

$$E(\widetilde{m}) \leq n \left(\sum_{d=1}^{k} \frac{1}{d} + k \sum_{d=k+1}^{\infty} \frac{1}{d^2} \right) < nH_k + nk\frac{\pi^2}{6}$$

Lower bound ($\gamma = 3$)

$$E(\widetilde{m}) \geq n \left(\sum_{d=1}^{k} \frac{1}{d^2} + k \sum_{d=k+1}^{\infty} \frac{1}{d^3} \right) > n\zeta(3)$$

Where $\zeta(3)$ is the Apéry's constant.
A closed form for the number (in expectation) of the lost edges is:

$$n \sum_{d=k+1}^{\infty} \frac{(d-k)}{d^\gamma}$$

Which can be upper-bounded:

$$n \sum_{d=k+1}^{\infty} \frac{(d-k)}{d^\gamma} < n \sum_{d=k+1}^{\infty} \frac{d}{d^\gamma} < n \sum_{d=1}^{\infty} \frac{d}{d^\gamma} < n\zeta(\gamma - 1)$$

4.2.2 Degree distribution

For all the nodes with degree less than k the distribution is preserved as in the original graph; instead, every node with degree $> k$ will be reduced as a node of k edges. Hence, the sample graph will have only nodes with at most k outgoing edges.
As we said, the effectiveness of our sample depends on the application, but in very large networks, such as the scale-free ones, this is generally bad; if k is too low, we might lose a lot of sensible informations of the network. For example, for the big "hubs" we might lose a lot of outgoing edges, which causes a lost of connections between the so-called "communities" (just think to the social graph of "Facebook"), an average path length shorter than the original graph, and the well-known "small-world phenomenon" might be affected due to our approximation.
On the other hand, all "small" vertices (the vast majority of the nodes) having a degree lower than k are not affected by our sampling operation;

hence, it is likely that all the local property of the network will be preserved. Here we analyze the degree distribution of both the network models.

Random graph: for the random graph, the out/in-degree distribution is the following:

$$P(h) = \binom{n-1}{h} p^h (1-p)^{n-1-h}$$

Where h is the exact number of connection for the nodes.

The distribution for the sample graph is different because we do not have any node with more than k connections, and nodes that in the original graph have more than k outgoing edges become nodes with exactly k outgoing edges. Hence the distribution is the same for $h < k$, whereas all the probability mass for $h \geq k$ collapse in $h = k$. More precisely:

$$\widetilde{P}_{out}(h) = \begin{cases} \binom{n-1}{h} p^h (1-p)^{n-1-h} & \text{for } 0 \leq h < k \\ 1 - \binom{n-1}{h} p^h (1-p)^{n-1-h} & \text{for } h = k \\ 0 & \text{for } h > k \end{cases}$$

For $n \to \infty$ the Binomial distribution approaches to a Poisson distribution, with a shorter tail w.r.t. the scale-free network; this means that the random graph just described is less affected than the scale-free network.

Scale-free network: by definition, a scale-free network has a in/out degree distribution which follows the power law, defined in (1).

For the out-degree distribution: the sample graph from this kind of network will have no node with degree more than k and, in the same fashion as in the case of random graph, we will collapse the distribution for degrees higher than k:

$$\widetilde{P}_{out}(h) = \begin{cases} \frac{1}{h^\gamma} & \text{for } 0 < h < k \\ 1 - \sum_{d=0}^{k} d^{-\gamma} & \text{for } h = k \\ 0 & \text{for } h > k \end{cases}$$

For the in-degree distribution: there is no easy prediction about how can be affected, since the dropping of an edge depends only from the properties of the source node.

4.2.3 Clustering coefficient

The *local clustering coefficient for a directed graph* of a node i is:

$$C_i = \frac{|\{e_{jk} : v_j, v_k \in N_i, e_{jk} \in E\}|}{k_i(k_i - 1)}$$

Where:

- N_i is the neighborhood of node i, defined as follows:

$$N_i = \{v_j : e_{ij} \in E \vee e_{ji} \in E\}$$

- $k_i = |N_i|$

In words, it is the number of connection between two neighbors of a node i over the number of all the possible connections between the neighbors. The *network average clustering coefficient* is simply the average of the local clustering coefficients defined above:

$$\bar{C} = \frac{1}{n} \sum_{i=1}^{n} C_i$$

In our sample, since we have less edges per node, the clustering coefficient will be lower than the original graph. For "small" clusters (i.e. the average degree is less than the threshold k) the clustering coefficient will be preserved; moreover, even if we lost a small number of edges, using some technique of link prediction it may be possible to recover some dropped link. Since we are dealing with directed graph, and due to our definition of neighborhood, it will be useful define some entities:

- N_i^{in}: set of nodes that are neighbors of i because it has *only* in-edges from them;
- N_i^{out}: set of nodes that are neighbors of i because it has *only* out-edges to them;
- $N_i^{in,out}$: set of nodes that are neighbors of i because it has *both* out-edges and in-edges to/from them;

Obviously, these sets are disjoint and the following holds: $N_i = N_i^{in} \cup N_i^{out} \cup N_i^{in,out}$

General analysis: Intuitively, the clustering coefficient is not affected if refers to nodes which have less than k outgoing edges and each node of their neighborhood has still less than k outgoing edges. The reason is simple: no edges is dropped in this subgraph, then for our central node the clustering coefficient is not affected.

More in general, we have several causes for which the local clustering coefficient for a generic node i might decrease:

- i has more than k out-going edges. This means that, uniformly, $|E_i|-k$ neighbors in N_i^{out} will be excluded from $\widehat{N}_i^{out}$ and hence the clustering coefficient will be affected only if these nodes have some connections between the other neighbors.

- Some of the nodes in N_i^{in} might have more than k out-going edges; hence, if for some of these nodes we drop the edge which makes them neighbor of i, then they will be excluded from $\widehat{N}_i^{in}$ and hence the clustering coefficient will be affected only if these nodes have some connections between the other neighbors.

- Both previous scenarios combined, where turns out that, due to edge deletions, a neighbor will be excluded from $\widehat{N}_i^{in,out}$.

Due to the complexity of the scenarios, it is very hard to find an elegant and useful generalized computation of the clustering coefficient for our sample.

5 Problem 5

5.1 Brief Summary

All the needed details for run the software are in the `README` file. The output produced by the software is located in the path specified as parameter when you run the program.
For each task, the relative output is in

`<outdir>/<TaskClassname>/(<day>|<startweek-endweek>).txt`

Now I show a brief summary for each assigned task.

- Task 1: top-5 countries per hour, per day.
  The class that implements this task is `batch.BatchTop5Country`.
  The output format is:

  ```
  (COUNTRY_CODE_1,NUM_OCC)
  (COUNTRY_CODE_2,NUM_OCC)
  ...
  ```

  Where `COUNTRY_CODE` is the code of the country in format , taken from the field `ActionGeo_CountryCode` (FIPS 10-4 country codes standard). E.g.:

  ```
  From out/byDay/20130401.txt:
  (US,8068)
  (AG,1554)
  (IN,1189)
  (UK,760)
  (CA,629)

  From out/byWeek/20130401-20130407.txt:
  (US,51638)
  (UK,7777)
  (IN,7050)
  (IS,6773)
  (CH,5542)
  ```

- Task 2: top-5 events per hour, per day
  The class that implements this task is `batch.BatchTop5Event`.
  The output format is:

```
(EVENT_CODE_1,NUM_OCC)
(EVENT_CODE_2,NUM_OCC)
...
```

Where `EVENT_CODE` is the code of the event taken from the field `EventBaseCode`. This field represent the code of the event, following the CAMEO taxonomy (second level of the tree). E.g.:

```
From out/byDay/20130401.txt:
(010,2315)
(042,1944)
(043,1805)
(040,1610)
(020,1460)

From out/byWeek/20130401-20130407.txt:
(010,17231)
(042,13848)
(043,12849)
(040,12289)
(020,10548)
```

- Task 3: top-5 leaders mentioned per hour, per day
  The class that implements this task is `batch.BatchTop5Leader`.

  The output format is:

  ```
  (LEADER_NAME_1,NUM_MENTIONS)
  (LEADER_NAME_2,NUM_MENTIONS)
  ...
  ```

  Where `LEADER_CODE` is the name of the identified leader. The extraction implementation is in `utils.LeaderExtractor`. I pick both Actor for each event and filter the result by "Actor(1|2)Type1Code" (aka Primary Role Code) that should be in some categories. Then I filter out the Actors who have some name in the dictionary which we do not want (e.g. "TERRORIST", "PRIME MINISTER"), which says anything about the actual leader.
  E.g.:

```
From out/byDay/20130401.txt:
(UNITED STATES,2850)
(OBAMA,2063)
(HAMAS,894)
(THE WHITE HOUSE,431)
(JACOB ZUMA,411)

From out/byWeek/20130401-20130407.txt:
(OBAMA,20750)
(UNITED STATES,18896)
(NASA,5897)
(BARACK OBAMA,4768)
(HAMAS,3839)
```

Notice: I preferred to do not reject results such as "UNITED STATES" and "HAMAS", because maybe they can be linked to actual leader. The results can be improved a lot, maybe considering some other source/tool: just think some dictionary, which would filter all the common nouns, or some named entity recognizer.

- Task 4: trending leaders (fast increasing number of mentions) per hour, per day
  The class that implements this task is `batch.BatchTrendingLeader`. The output format is:

  ```
  (LEADER_NAME,DELTA_NUM_MENTIONS)
  (LEADER_NAME,DELTA_NUM_MENTIONS)
  ...
  ```

  Where `DELTA_NUM_MENTIONS` is the difference of the total number of mentions between th current window and the last one. E.g.:

  ```
  From out/byDay/20130402.txt:
  (OBAMA,1202)
  (NASA,670)
  (BARACK OBAMA,563)
  (HEALTH OFFICIAL,533)
  (UNITED STATES,425)
  (MALAYSIA,362)
  (INTELLIGENCE,312)
  ```

```
(CENTRAL BANK,296)
(MARINES,273)
(CHINA,235)

From out/byWeek/20130409-20130415.txt:
(UNITED STATES,10637)
(INTELLIGENCE,3070)
(GERMANY,1410)
(BARACK OBAMA,1343)
(SOUTH SUDAN,1278)
(US OFFICIAL,1103)
(THE WHITE HOUSE,1050)
(CAPTAIN,1042)
(LI KEQIANG,1000)
(APPEALS COURT,964)
```

- Task 5: violent events per hour, per day
  The class that implements this task is `batch.BatchViolentEvent`.

  The output format is:

  ```
  (VIOLENT_EVENT_CODE,NUM_OCC)
  (VIOLENT_EVENT_CODE,NUM_OCC)
  ...
  ```

  Where `VIOLENT_EVENT_CODE` is the code of the event considered violent. The implementation of the extractor is in `utils.ViolentEventExtractor`. It filters events based on some type of event categories. For example, code "19", in the CAMEO standard, stands for the category "FIGHT"[7], while "18" for "ASSAULT"[8]. E.g.:

  ```
  From out/byDay/20130401.txt:
  (19,2014)
  (18,560)
  (145,36)
  (20,12)
  ```

[7] "Use conventional military force"
[8] "Use unconventional violence."

```
(175,3)
```

```
From out/byWeek/20130401-20130407.txt:
(19,14340)
(18,3346)
(145,171)
(175,150)
(20,69)
```

- Task 6: protests per hour, per day
  The class that implements this task is `batch.BatchProtestEvent`.

  The output format is:

  ```
  (PROTEST_EVENT_CODE,NUM_OCC)
  ```

  Where `PROTEST_EVENT_CODE` is the code of the event considered as a protest. The implementation of the extractor is in `utils.ProtestEventExtractor`. It filters events based on the event root code "14". E.g.:

  ```
  From out/byDay/20130401.txt:
  (14,258)
  ```

  ```
  From out/byWeek/20130401-20130407.txt:
  (14,1958)
  ```

- Task 7: violent vs non-violent actors per hour, per day
  The class that implements this task is `batch.BatchViolentNonViolentActorsMain`.

  The output format is:

  ```
  (GLOBAL_EVENT_ID,ACTOR1,ACTOR2)
  ...
  ```

  Where `GLOBAL_EVENT_ID` is the unique id of the event considered as a violent vs non-violent event. ACTOR1 and ACTOR2 are the full names. The implementation of the extractor is in `batch.BatchViolentNonViolentActorsMain.ViolentNonViolentExtractor`. It filters events based on the event root code "18" (category "ASSAULT"). E.g.:

```
From out/byDay/20130401.txt:
(253465172,AFGHAN,PERTH)
(253465183,KANDAHAR,SECURITY FORCE)
(253465229,AFGHAN,HINDU)
(253465253,AFGHAN,ISRAEL)
(253465268,AFGHAN,MILITARY OFFICIAL)
...
```

- Task 8: state vs non-state actors per hour, per day
  The class that implements this task is `batch.BatchStateNonStateActorsMain`.

  The output format is:

  ```
  (GLOBAL_EVENT_ID,ACTOR1,ACTOR2)
  ...
  ```

  As before, but ACTOR1 is a "state actor", while ACTOR2 is a "non-state actor". The implementation of the extractor is in `batch.BatchStateNonStateActorsMain.StateNonStateExtractor`. It filters events based on the "Actor1Type1Code", that should be "GOV", and "Actor2Type1Code", that should be not equal to "GOV". E.g.:

  ```
  From out/byDay/20130401.txt:
  (253465418,HAMID KARZAI,TALIBAN )
  (253465419,HAMID KARZAI,TALIBAN )
  (253465420,HAMID KARZAI,SMUGGLER)
  (253465421,HAMID KARZAI,SMUGGLER)
  (253465422,AFGHANISTAN,ENGINEER)
  (253465423,AFGHAN,ENGINEER)
  (253465424,AFGHAN,CIVILIAN)
  (253465427,AFGHAN,MILITARY)
  (253465428,AFGHANISTAN,ISLAMIC)
  (253465442,KANDAHAR,KANDAHAR)
  (253465666,AFRICA ,GOVERNMENT FORCES)
  ```

- Task 9: trending violent regions (fast increasing number of events) per hour, per day.
  The class that implements this task is `batch.BatchTrendingCountryByViolentEvent`.

  The output format is:

```
(COUNTRY_CODE,DELTA_NUM_VIOLENT_EVENTS)
...
```

Where `DELTA_NUM_VIOLENT_EVENTS` is the difference on the number of violent events by country between the current window values and the ones of the previous window. E.g.:

```
From out/byDay/20130402.txt:
(IS,181)
(LY,91)
(PK,52)
(SF,41)
(SO,31)
(BR,31)
(UK,26)
(KN,22)
(IT,15)
(EI,14)

From out/byWeek/20130409-20130415.txt:
(US,990)
(RS,632)
(SY,564)
(PK,377)
(SU,209)
(OD,179)
(SO,167)
(RI,140)
(RB,116)
(LE,110)
```

5.2 Other informations

The experiment was executed over a single computer with 16 GB of RAM memory.
The software, for simplicity, has been implemented to process .csv files per day; so I started from the available dataset in this format. However, it is easily extensible to partition the data by date.
Due to scarcity of time and disk resources, I analyzed only the year 2013.
For a single year and a single task, it took no more than 5 minutes.

References

[1] Noga Alon, Yossi Matias, and Mario Szegedy. "The Space Complexity of Approximating the Frequency Moments". In: *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*. STOC '96. Philadelphia, Pennsylvania, USA: ACM, 1996, pp. 20–29. ISBN: 0-89791-785-5. DOI: 10.1145/237814.237823. URL: http://doi.acm.org/10.1145/237814.237823.

[2] Noga Alon and Joel H. Spencer. *The Probabilistic Method*. 4th. Wiley Publishing, 2016. ISBN: 1119061954, 9781119061953.

[3] Herman Chernoff. "A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the sum of Observations". In: *Ann. Math. Statist.* 23.4 (Dec. 1952), pp. 493–507. DOI: 10.1214/aoms/1177729330. URL: http://dx.doi.org/10.1214/aoms/1177729330.

[4] Chris Ding and Xiaofeng He. "K-means Clustering via Principal Component Analysis". In: *Proceedings of the Twenty-first International Conference on Machine Learning*. ICML '04. Banff, Alberta, Canada: ACM, 2004, pp. 29–. ISBN: 1-58113-838-5. DOI: 10.1145/1015330.1015408. URL: http://doi.acm.org/10.1145/1015330.1015408.

[5] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. "CURE: An Efficient Clustering Algorithm for Large Databases". In: *SIGMOD Rec.* 27.2 (June 1998), pp. 73–84. ISSN: 0163-5808. DOI: 10.1145/276305.276312. URL: http://doi.acm.org/10.1145/276305.276312.

[6] Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets*. New York, NY, USA: Cambridge University Press, 2011. ISBN: 1107015359, 9781107015357.

[7] Jeffrey S. Vitter. "Random Sampling with a Reservoir". In: *ACM Trans. Math. Softw.* 11.1 (Mar. 1985), pp. 37–57. ISSN: 0098-3500. DOI: 10.1145/3147.3165. URL: http://doi.acm.org/10.1145/3147.3165.

[8] Qin Xu et al. "PCA-guided search for K-means". In: *Pattern Recognition Letters* 54 (Mar. 2015), pp. 50–55. ISSN: 0167-8655. DOI: 10.1016/j.patrec.2014.11.017.

[9] Hongyuan Zha et al. "Spectral Relaxation for K-means Clustering". In: *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*. NIPS'01. Vancouver, British Columbia, Canada: MIT Press, 2001, pp. 1057–1064. URL: http://dl.acm.org/citation.cfm?id=2980539.2980675.