Parallel Sparse K-Means for Document Clustering

Victor Storchan, CME 323 (R. Zadeh)

June, 2016

Abstract

Through this paper, a Parallel Sparse K-Means procedure is implemented with a particular care on improving the scalability and the efficiency of the regular K-Means when one wants to apply it to sparse data sets. As an example, the method is implemented and tested on the 20 Newsgroups data set. All the code used in the paper can be found at `https://github.com/victorstorchan/doc_clustering`

Contents

1 Introduction and related work **2**

2 K-Means as a maximisation problem **2**

2.1 Some measures to set the problem 2

2.2 The additivity of features's dissimilarities 4

2.3 Depth and Work for SK-Means algorithm 5

3 K-Means with a K-Means parallel initialization **6**

3.1 Deriving K-Means|| 6

3.2 Depth and work for K-Means|| 7

3.3 The K-Means algorithm with parallel initialization 7

3.4 Depth and work for K-Means with parallel initialization 8

4 The Parallel Sparse K-means algorithm **8**

5 PSK algorithm on a cluster **8**

6 Running the algorithm on the 20 Newsgroups Data Set **9**

6.1 Description of the Data Set 9

6.2 Cleaning the Data Set 9

6.3 The TF-IDF model 10

6.4 Running Parallel Sparse K-Means 11

7 Conclusion **13**

8 References **13**

1 Introduction and related work

The partitioning of data is a problem of combinatorial optimization. The problem is to divide N points of a p dimensional space into K clusters by minimizing a cost function. The distance of a point to the mean of the points of its cluster will be considered. In particular, document clustering can be viewed as an application of this method. Given a bunch of documents we want to be able to recover common patterns among them, based on the words they contain. The different labels are the topics of the documents (recreational, talk, science etc..). To transform the different documents into points of a p dimensional space, we will use the famous TF-IDF model is used from MLlib and implemented with the hashing trick. Rather than updating a dictionary, a vector of a pre-defined length (here, $p = 2^{16}$) is built by applying a hash function h to the words (features) of the documents. Then using the hash values as indices, the related values at those indices is updated. The **clustering** package of MLlib contains in particular, a list of non-supervised algorithms for which the goal is not to predict a variable, but to gather the data points into clusters with strong similarity. The framework of the work is the following: our goal is to efficiently classify a bunch of N documents . $N = 11314$. By the term "efficiently", what we mean here is "quickly and accurately". To this purpose, we will develop an algorithm (PSK-Algorithm) which takes into account both the sparsity of the data points inerent to a TF-IDF model (section 2), but also which takes into account the quantity of the data to address the scalability of our algorithm (sections 3 and 4). As a consequence, a parallel initialisation of K-Means will be plugged into a Sparse version of K-Means. Although at some point in the PSK-algorithm, the classical K-Means has to be run in paralell, this problem is embarrassingly parallel and we will not focus too much on this part, as it is not of critical need for our numerical results. We only give an overview of the implementation in MapReduce in section 3. To run the experiment, we will use the 20 Newsgroups data set in section 5. As the Sparse K-means algorithm is described in [2] and the parallel initialization for K-Means is described in [1], the novelty of the work is to merge these two parts and to parallelize the resulting algorithm as well as we can, to implement a procedure which is scalable, fast, and specific for sparse data.

2 K-Means as a maximisation problem

2.1 Some measures to set the problem

First, let's define a couple of measures. The definition basically states what quantity is measured.

- **The overall between cluster dissimilarity:**

$$OBCD(\mathcal{C}) = \sum_{k=1}^{K} \sum_{i \in C_k} \sum_{i' \notin C_k} d(x_i, x_{i'})$$

- **The overall total cluster dissimilarity:**

$$OTCD = \sum_{i=1}^{N} \sum_{i'=1}^{N} d(x_i, x_{i'})$$

  It is the sum of dissimilarities of all pairs, so it does not depend on $\mathcal{C}$.

- **The overall within cluster dissimilarity:**

$$OWCD(\mathcal{C}) = \sum_{k=1}^{K} \sum_{i,i' \in C_k} d(x_i, x_{i'})$$

Prop: A simple computation shows $OWCD(\mathcal{C}) = OTCD - OWCD(\mathcal{C})$. Let $|C_k| = n_k$. To avoid the case where $n_1 = n_2 = ... = n_K$ while minimizing $OWCD$ with respect to $\mathcal{C}$ we now define:

- **The within cluster dissimilarity:**

$$WCD(\mathcal{C}) = \sum_{k=1}^{K} \frac{1}{2n_k} \sum_{i,i' \in C_k} d(x_i, x_{i'})$$

  Then the goal of K-means is to find an optimizing partition of centers $\mathcal{C}^*$ such that

$$\mathcal{C}^* = argmin_{\mathcal{C}:|\mathcal{C}|=K} WCD(\mathcal{C})$$

  Let's mention here the particular case of interest when d is the L^2 distance: then, we will define

- **The within cluster sum of squares:**

$$WSS(\mathcal{C}) = \sum_{k=1}^{K} \frac{1}{2n_k} \sum_{i,i' \in C_k} \|x_i - x_{i'}\|^2$$

  Acomputation gives:

$$WSS(\mathcal{C}) = \sum_{k=1}^{K} \sum_{i \in C_k} \|x_i - \overline{x_k}\| \text{ where } \overline{x_k} \text{ is the cluster mean for cluster } C_k.$$

- **The total cluster dissimilarity:**

$$TCD = \frac{1}{2N} \sum_{i=1}^{N} \sum_{i'=1}^{N} d(x_i, x_{i'})$$

  Then,

$$BCD(\mathcal{C}) = TCD - WCD(\mathcal{C})$$

We deduce from these definitions that:

$$\begin{aligned} \mathcal{C}^* = argmin_{\mathcal{C}:|\mathcal{C}|=K} WCD(\mathcal{C}) &= argmin_{\mathcal{C}:|\mathcal{C}|=K} (TCD - BCD(\mathcal{C}) \\ &= argmin_{\mathcal{C}:|\mathcal{C}|=K} (-BCD(\mathcal{C})) \\ &= argmax_{\mathcal{C}:|\mathcal{C}|=K} BCD(\mathcal{C}) \end{aligned}$$

Eventually:

- **The total sum of squares:**

$$TSS = \frac{1}{2N} \sum_{i=1}^{N} \sum i' = 1^N \|x_i - x_{i'}\|^2$$

- **The between sum of squares:**

$$BSS(\mathcal{C}) = \sum_{i=1}^{N} \|x_i - \overline{x}\|^2 - \sum_{k=1}^{K} \sum_{i \in C_k} \|x_i - \overline{x_k}\|^2$$

Then, $TSS = BSS(\mathcal{C}) + WSS(\mathcal{C})$ and

$$\mathcal{C}^* = argmax_{\mathcal{C}:|\mathcal{C}|=K} BSS(\mathcal{C})$$

2.2 The additivity of features's dissimilarities

Without loss of generality, we will make the following assumptions for our distance: let $d_j(x_i, x_{i'})$ be the distance (or dissimilarity) between points x_i and $x_{i'}$ with respect to the j^{th} feature. We suppose that $d(x_i, x_{i'}) = \sum_{j=1}^{p} d_j(x_i, x_{i'})$.This assumptions is verified when d is the distance derived from the L^2 norm (the distance of interest). As a consequence, a simple computation shows that:

$$WCD(\mathcal{C}) = \sum_{j=1}^{p} WCD_j(\mathcal{C})$$

$$TCD(\mathcal{C}) = \sum_{j=1}^{p} TCD_j(\mathcal{C})$$

$$BCD(\mathcal{C}) = \sum_{j=1}^{p} BCD_j(\mathcal{C})$$

where WCD_j, TCD_j and BCD_j only depend on the j^{th} feature (ie j^{th} column of the matrix of predictors X). As a consequence, the idea is that $BSS_j(\mathcal{C})$ can be seen as the contribution of the j^{th} feature to the clustering. Thus, we will divide the set of features into two subsets: the clustering features which have large values of BSS_j and the noise features which have small values of BSS_j. Physically, a noise feature can be thought as a feature that does not change from cluster to cluster. In the NLP case, it can be a word whose frequency does not vary much from a cluster to another. We are now ready to derive the Sparse K-means problem as a convex problem: consider the following problem:

$$max_{|\mathcal{C}|=K,w} \sum_{j=1}^{p} w_j BSS_j(\mathcal{C})$$

$$\text{such that} \begin{cases} \|w\|_1 \leq l_1 \\ \|w\|_2^2 \leq 1 \\ w_j \geq 0 \end{cases}$$

The idea is that only clusturing features will receive nonzero weights. Moreover, the sparsity of w is garanteed for small values of the parameter $l_1 \in \{1, ..., \sqrt{p}\}$. Without the L^2 condition, we can show that at most one element of w would be nonzero (the w_j relative to the biggest clustering feature). Of course, one wants to avoid this case.
Now let's derive an algorithm to solve this particular relaxed convex problem. Consider the following procedure: The analytical solution of the problem in line 6 of algorithm 1 is the famous water-filling described in the Boyd on page 245 (see [8]), derived from the KKT conditions. For the sake of clarity, let state what is the solution $w(\Delta^*)$ of the problem:

$$w(\Delta) = \frac{(D - \Delta\mathbf{1})_+}{\|(D - \Delta\mathbf{1})_+\|_2}$$

where

$$\Delta^* = \begin{cases} 0 \text{ if } \|w(0)\|_1 \leq l_1 \\ \text{Solution of } \|w(\Delta)\|_1 = l_1 \text{ otherwise} \end{cases}$$

Algorithm 1 $SK - Means(K)$

```
1: procedure SK-MEANS
2:     $w \leftarrow \frac{1}{p}[1, 1, ..., 1]^T$
3:     Repeat (until convergence):
4:         Step (a):
5:         $Y := [\sqrt{w_1}X_1, ..., \sqrt{w_p}X_p]$
6:         for a given $w$, perform K-means on the transformed data set Y:
7:         solve $\tilde{\mathcal{C}} = argmin_{|\mathcal{C}|=K} WSS_y(\mathcal{C})$ where $WSS_y(\mathcal{C})$ is the within cluster sum of squares
8:         on Y.
9:         Step (b):
10:        For a given $\tilde{\mathcal{C}}$, maximize with respect to $w$: set $D = [BSS_1(\tilde{\mathcal{C}}), ..., BSS_p(\tilde{\mathcal{C}})]$ and
11:        solve:
```

$$\tilde{w} = max_w w^T D$$

$$\text{such that} \begin{cases} \|w\|_1 \leq l_1 \\ \|w\|_2^2 \leq 1 \\ w_j \geq 0 \end{cases}$$

As $\|w(\Delta)\|_1$ is a piecewise linear and increasing function, the equation above has a unique solution.
Remark: Let's recall that the goal of SK-Means is to find

$$argmax_{|C|=K} \sum_{j=1}^{p} w_j BSS_j(\mathcal{C}) = argmin_{|C|=K} \sum_{j=1}^{p} w_j WSS_j(\mathcal{C}).$$

Or a simple computation implies that $WSS(\mathcal{C}) = \sum_{j=1}^{p} w_j WSS_j(\mathcal{C})$. It is directly derived from the definitions. As a consequence, SK-Means tries to recover $argmin_{|\mathcal{C}|=K} WSS_Y(\mathcal{C})$ Moreover, if $l_1 \leq 1$, then the L^2 constraint is redundant and in that case, $w = l_1 e_{j_0}$ where $j_0 = argmax_j BSS_j(\mathcal{C})$ ie 100% of the weight will be put on the feature with the largest $BSS_j(\mathcal{C})$. On the other hand, if $l_1 > \sqrt{p}$ then, the L^1 constraint is redundant and all the features will have nonzero weights. As a consequence, depending on the "degree of sparsity" we will choose $l_1 \in [1, .., \sqrt{p}]$.

2.3 Depth and Work for SK-Means algorithm

On a PRAM model, let's first compute the time complexity of part (a) of SK-Means:

- work of step (a): $W(N, K, p) = W_{K-Means\ parallel} + W_{transformX \to Y}$
- Depth of step (b): $D(N, K, p) = D_{K-Means\ parallel} + D_{transformX \to Y}$ where $D_{transformX \to Y} = O(1)$.

The time complexity for part (b) is the following:

- work of step (b): computing Δ^*: Solving $\|w(\Delta)\|_1 = l_1$ is solving $\sum_{j=1}^{p} w_i = l_1$, so it is setting $w_p = l_1 - \sum_{j=1}^{p-1} w_i$. So using prefix sum, the work is $O(p)$. Then computing D is $O(Np)$ work and computing $w(\Delta)$ is another $O(p)$.

- Depth of step (b): computing Δ^*: with prefix-sum, the depth is $O(log(p))$. Then computing D is $O(log(N) + log(p))$. Computing $w(\Delta)$ is $O(1)$ depth.

3 K-Means with a K-Means parallel initialization

3.1 Deriving K-Means||

For a point $x \in X$ and a subset $Y \subset X$, we adopt the following notations:

- $d(x, Y) = min_{y \in Y} \|x - y\|_2$
- $mean(Y) = \frac{1}{|Y|} \sum_{y \in Y} y$
- $\phi_Y(\mathcal{C}) = \sum_{y \in Y} d^2(y, \mathcal{C}) = \sum_{y \in Y} min_{i=1,.,K} \|y - c_i\|^2$ is the cost function of Y.

The goal of K-Means, can be formulated as minimizing $\phi_X(\mathcal{C})$ over $\mathcal{C}$. Let's call the optimal solution $\mathcal{C}^*$. Note that finding $\phi^* = \phi_X(\mathcal{C}^*)$ is NP-hard. We say that $\mathcal{C}$ is an α-approximation of $\mathcal{C}^*$ if $\phi_X(\mathcal{C}) \leq \alpha\phi^*$. It is worth to note that any centers set $\mathcal{C}$ can be identify with a clustering: $x \in X$ is in C_j if $j = argmin_i \|x - c_i\|$. In order to see the real improvment of the K-Means|| algorithm, let's first derive quickly the sequential K-Means++ algorithm, which provides a good initialization step but which can not be parallelized. It is a $8log(K)$-approximation algorithm.

Algorithm 2 $K - Means + +$

```
1: procedure K-MEANS++
2:     $\mathcal{C} \leftarrow$ Sample one point uniformly at random in X
3:     while $|\mathcal{C}| < K$ do
4:         Sample $x \in X$ with probability $\frac{d^2(x,\mathcal{C})}{\phi_X(\mathcal{C})}$
5:         $\mathcal{C} \leftarrow \mathcal{C} \cup \{x\}$
```

The density of probability in K-Means++ is quantifying how far the point x is from the previously selected center. It has as many chances to get selected as it is far from them. The density is renormalized to sum to 1. As K-Means++ is sequential by nature, we follow the idea of [1] to implement the K-Means|| algorithm for a good initialization of the clusters. This algorithm will then be plugged in the initialization step of the SK-Mean algorithm. An oversampling factor l is needed. One can think about l as $\Theta(K)$.

Algorithm 3 $K - Means||$

```
1: procedure K-MEANS||
2:     $\tilde{\mathcal{C}} \leftarrow$ Sample one point uniformly at random in X
3:     $\psi \leftarrow \phi_X(\tilde{\mathcal{C}})$
4:     for $O(log(\psi)$ times do
5:         $C' \leftarrow$ Sample each $x \in X$ independently with probability $\frac{ld^2(x,\tilde{\mathcal{C}})}{\phi_X(\tilde{\mathcal{C}})}$
6:         $\tilde{\mathcal{C}} \leftarrow \tilde{\mathcal{C}} \cup C'$
7:     for $x \in \tilde{\mathcal{C}}$ do
8:         set $w_x = \#$ { points in $X$ closer to $x$ than any other points in $\tilde{\mathcal{C}}$ }
9:     Recluster the weighted points in $\tilde{\mathcal{C}}$ into $K$ clusters: use K-Means++ on
10:    $\{w_1 c_1 \in R^p, ..., w_{|\tilde{\mathcal{C}}|} c_{|\tilde{\mathcal{C}}|} \in R^p\}$. Return $\mathcal{C}$
```

Since $|\mathcal{C}| \ll N$, ie the number of clusters K is supposed significantly smaller than the data size, we suppose that reclustering on a single machine using K-Means++ is possible and fast.

Theorem: If an α-approximation algorithm is used for the reclustering on a single machine, then, K-Means|| is an $O(\alpha)$-approximation algorithm to K-Means. In particular if K-Means++ is used, then $\alpha = log(K)$.

3.2 Depth and work for K-Means||

Suppose that concurrent write is legitimate. Let's now derive the work and the depth for algorithm 3 (K-Means||):

- Line 2: $D(N, \tilde{\mathcal{C}}, p) = W(N, \tilde{\mathcal{C}}, p) = O(1)$
- Line 3: computing $\phi_X(\tilde{\mathcal{C}})$, $W(N, K, p) = O(|\tilde{\mathcal{C}}|^2 * N^2 * p)$ (p for the dot product, K for taking the minimum over a set of length $|\tilde{\mathcal{C}}|$, and N to compute the sum. As we need to do it for each tuple $(c_i, x) \in \tilde{\mathcal{C}} * X$, we squared the relevant quantities.
  The depth is $D(N, \tilde{\mathcal{C}}, p) = O\big(log(p) + log(\tilde{\mathcal{C}}) + log(N)\big)$
- Line 5: Need to compute $\frac{ld^2(x, \tilde{\mathcal{C}})}{\phi_X(\tilde{\mathcal{C}})}$: We can store from the previous step the denominateur. So one just needs to compute the numerator: $W(N, \tilde{\mathcal{C}}, p) = O(|\tilde{\mathcal{C}}| * p)$
  $D(N, \tilde{\mathcal{C}}, p) = O(log(p) + log(|\tilde{\mathcal{C}}|)$
- Line 9: Depth: depth of K-Means++.
  Work: work of K-Means++ plus $|\tilde{\mathcal{C}}|$ for weighting the centers.

Then let's derive the work and the depth of K-Means++:

- Line 4: Depth: $D(N, \tilde{\mathcal{C}}, p) = O\big(log(p) + log(\tilde{\mathcal{C}}) + log(N)\big) + O\big(log(p) + log(|\tilde{\mathcal{C}}|)\big)$
  $= O\big(log(p) + log(\tilde{\mathcal{C}}) + log(N)\big)$
  Work: $W(N, |\tilde{\mathcal{C}}|, p) = O\big(|\tilde{\mathcal{C}}|^2 * N^2 * p) + O\big(|\tilde{\mathcal{C}}| * p\big) = O\big(|\tilde{\mathcal{C}}|^2 * N^2 * p\big)$.

As line 5 has to be run $|\tilde{\mathcal{C}}|$ times, the total work for K-Means++ is $O\big(|\tilde{\mathcal{C}}|^3 * N^2 * p\big)$.

As a consequence, the total work of K-Means|| is $O\big(log(\psi) * |\tilde{\mathcal{C}}|^3 * N^2 * p\big)$ and the total depth for K-Means|| is $O\big(log(p) + log(|\tilde{\mathcal{C}}|) + log(N)\big)$.

3.3 The K-Means algorithm with parallel initialization

The idea is simple. Intuitively, the initialization exploits the fact that a good clusturing should be enough spread out: rather than choosing each time a new center at random, let's derive a good density of probability to choose a new cluster center with a pealty on the points which are too close from the previously selected centers. The well known $K - Means++$ algorithm is a $O(log(K)$-approximation algorithm of the optimum. It is even a constant approximation if the data is known well clusterable).

Remark: As it is mentionned in the introduction, the K-Means procedure is embarrassingly parallel, and implementing it with a MapReduce framework has already been studied a lot. As a reference, one can read [3] and [4]. The global idea is to do the classification during the map step by parallelizing over the data and computing the means during the reduce phase by parallelizing over the centers. From a practical point of view in python, this can be easily

Algorithm 4 $K - Means$ with parallel initialization

```
1: procedure K-Means with parallel initialization
2:     Initialize with K-Means||.
3:     Repeat (until convergence):
4:         Step (a):
5:         $\forall i$ set $c_i := argmin_j \|x^{(i)} - \mu_j\|^2$
6:         Step (b):
7:         $\forall j, \mu_j := \frac{\sum_{i=1}^{N} \mathbf{1}_{\{c^{(i)}=j\}} x^{(i)}}{\sum_{i=1}^{N} \mathbf{1}_{\{c^{(i)}=j\}}}$
```

done using the K-means function of **sklearn.cluster** module and specifying the parameter *n_jobs*=**-1** to use all the CPUs. The type of the matrix of prediction has then to be any sparse matrix class.

3.4 Depth and work for K-Means with parallel initialization

- Line 2: Work $W_{K-Means||}$ and depth $D_{K-Means||}$ of K-Means||, see subsection 3.2.
- Line 5: Work is $W(N, |\tilde{\mathcal{C}}|, p) = O\big(|\tilde{\mathcal{C}}| * p * N\big)$ and depth is $D(N, |\tilde{\mathcal{C}}|, p) = O\big(log(p) + log(|\tilde{\mathcal{C}}|)\big)$
- Line 7: $W(N, |\tilde{\mathcal{C}}|, p) = O(|\tilde{\mathcal{C}}| * N)$ and $D(N, |\tilde{\mathcal{C}}|, p) = O\big(log(N)\big)$

Let's suppose that we run $step(a)$ and $step(b)$ R times. Then the total work of K-Means is $O\big(W_{K-Means||} + R * |\tilde{\mathcal{C}}| * p * N\big)$.
The total depth is $O\big(D_{K-Means||} + R * (log(p) + log(|\tilde{\mathcal{C}}|) + log(N))\big)$.

4 The Parallel Sparse K-means algorithm

Now, we are ready to describe the PSK-Algorithm.

Algorithm 5 $PSK - Algorithm$

```
1: procedure PSK-Algorithm
2:     Run SK-Means with modified step (a):
3:         When performing K-means on the weighted data, initialize the procedure with
4:         K-Means|| and run the K-Means iterations in parallel using as many CPUs as
5:         available.
```

5 PSK algorithm on a cluster

Let's say, for this section that we are using a cluster composed of s machines connected to a master machine. Then, in a MapReduce framework, let's analyse first the communication cost for standard K-means: First, the master machine reads the clustering center files and then shuffles it to the machines. Then, each $mapper_i$ for $i \in \{1, ..., s\}$ assigns a cluster center for each points $x \in X_i \subset X$ and record the local minimum distance $d(c_i, X_i)$. It results in s vectors of length k that have to be reduced (the reducers compute the global sums, update the centers and send this

data to all the processors with an $MPI\_AllRedude$ procedure for example). As the local min are vectors of length K for every s machines, the communication cost is $O(sk)$. The reduce-key space is $\frac{N}{s}$.
Let's now look at the MapReduce implementation of the initialization step $k-means||$, assuming that $|\tilde{\mathcal{C}}|$ is fitting in memory. On line 5 of algorithm 3, each mappers sample in their subsapces X_i in parallel. They are also computing $\phi_{X_i}(\tilde{\mathcal{C}})$ and the reducer adds these values to compute the global $\phi_X(\tilde{\mathcal{C}})$. The shuffle cost time is her $O(s)$.

6 Running the algorithm on the 20 Newsgroups Data Set

For this section, all the code which was used to produce the different results can be found at the following url: `https://github.com/victorstorchan/doc_clustering`. The prerequisite to run the different part of the pipeline are: Pyspark, SparkR, Sparcl (where SK-Means is implemented), and rPython (to shuffle data from python to R).

6.1 Description of the Data Set

The 20 Newsgroups Data Set can be downloaded here: `http://qwone.com/~jason/20Newsgroups/`. The algorithm will be running on a subset of this Data Set, gathering 11314 emails. These emails are labeled into different classes and subclasses as follow:

computer	recreational	science	talk.politics
comp.graphics comp.os.ms-windows.misc comp.sys.ibm.pc.hardware comp.sys.mac.hardware comp.windows.x	rec.autos rec.motorcycles rec.sport.baseball rec.sport.hockey	sci.crypt sci.electronics sci.med sci.space	talk.politics.misc talk.politics.guns talk.politics.mideast

misc.forsale	religion
misc.forsale	talk.religion.misc alt.atheism soc.religion.christian

As a consequence, we can see how well our clustering algorithm is performing on one hand on the 7 "large" classes (comp, talk, rec, alt, soc, misc, and sci) and on the other hand on the 20 "more specific" classes. As it will be shown from the experiment later, we have to deal with a simple tradeoff: larger classes means a better overall accuracy but not a very precise clustering, and more specific classes means a better clustering with more chances to fail to predict the good specific label.

6.2 Cleaning the Data Set

As a reminder, all the code which has been used here is available at `https://github.com/victorstorchan/doc_clustering`. First, the data is put into an RDD vector. As a results the emails are split and distributed on the available network. Parsing the Data Set, is always the very first step of the work. To this purpose, we will remove the ponctuation, the figures, the capital letters, we will remove the stop words and the words which are present just once in all the emails (obviously, this kind of words will not give any information about a particular email). Moreover, we remove the words of length 1 and we return a tuple of the remaining words, and their count over all the emails. At the end of this parsing step, here is a quick view of the data.

Each element of the RDD (the former emails) is now a list of tuples of (words, count) where the count, is now the count of occurence of the word in a particular email (so it can be 1):
[('ink', 4), ('negative', 1), ('hand', 3), ('compared', 1), ('argues', 1), ('tendentious', 2), ('appendix', 2), ('laurel', 1), ('centuries', 1), ('goes', 1), ('kierkegaard', 1), ('atheisten', 2), ('deluxe', 1), ('pregnant', 1) etc...]
These counts are just here to give us an idea about which features would be relevant for classification, and which ones are not.

6.3 The TF-IDF model

From each email, we want to produce a vector based on the parsed information (ie remaining words) contained in the email. To this purpose, the famous TF-IDF model is used with a dimension of 2^{16}. This statistical measure is used to evaluate the significance of a word within a particular document comparatively to a collection or a corpus. To have a better insight on the method, one can find useful information at `https://en.wikipedia.org/wiki/Tf%E2%80%93idf`. The vectors are created with the modules *IDF* and *HashingTF* of *pyspark.mllib.feature*. Of course, as an email only contains a few words relative to an entire vocabulary, the resulting vectors are sparse vectors. We will exploit this fact while building the Parallel Sparse K-Means algorithm. These sparse vector are then stored on the SparseVector format of *pyspark.mllib.linalg*. Note that using a TF-IDF model is a bottleneck for streaming clustering of a huge set of documents because one needs to know the whole corpus vocabulary to implement TF-IDF. [5] gives a way for solving this issue. In practice, after the TF-IDF transformation, the dimension of the matrix of prediction is $1134 * 2^{16}$ so we can run a PCA. For this extra step of preprocessing, we did a *Truncated SVD* from the package *sklearn.decomposition*. However, using PCA has a number of disavantages: first of all, the resulting matrix of reduced dimension is not sparse anymore so we will not use the PSK algorithm on it. Additionally, there is no garantee that we will achieve best separation of clusters (ie minimum of the cost function) with this extra step. To conclude, as we stored a sparse representation of the matrix of prediction, it may not fit in memory. An other choice would be to run distribted PCA see [6].

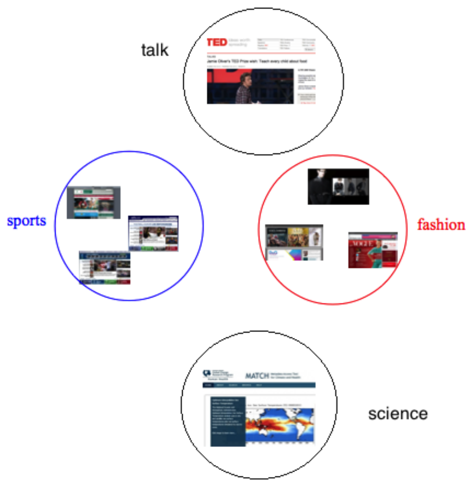
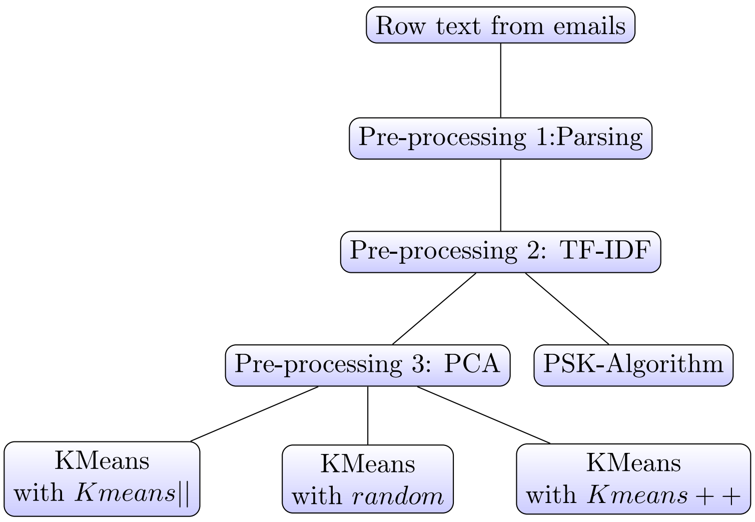


Figure 1: TF-IDF gives a spatial representation of the text data

6.4 Running Parallel Sparse K-Means

We are using several packages: First, the parallel initialization of PSK-algorithm is done with the implementation of MLlib. Then, we are shiffting all the data from python to R. Actually, to run the SK-Means, we will use the package **Sparcl** written in R by Witten and Tibshirani [2]. To shift the data, we shift the 11314 dictionaries corresponding to every emails by first saving them on the Hard Drive, and then loading them into R. All the code is written in python, and the library **rPython** is called from R to load python files from the R console. Indeed,it is too computationally heavy to first compute a dense matrix of prediction with pyspark then save it on the Hard Drive and shuffle it to the R console: The $(11314 * 2^{16})$ matrix takes 18Go of memory to be save as a ".txt" file for example. The graph below shows the whole pipeline used for this work. Notice that in practice, the PSK-Algorithm was trained on a subset of the 11314 samples because of computational time cost relative to loading the matrix of prediction in R with $rPython$.



Here are the numerical results that can be obtained: first we compare the results of the standard K-Means, with the several initializations, for the large classification ($K = 20$) and then, for the more specific one ($K = 7$). The accuracy is the rate of well labeled points. Moreover, the time for the procedure to finish is also recorded:

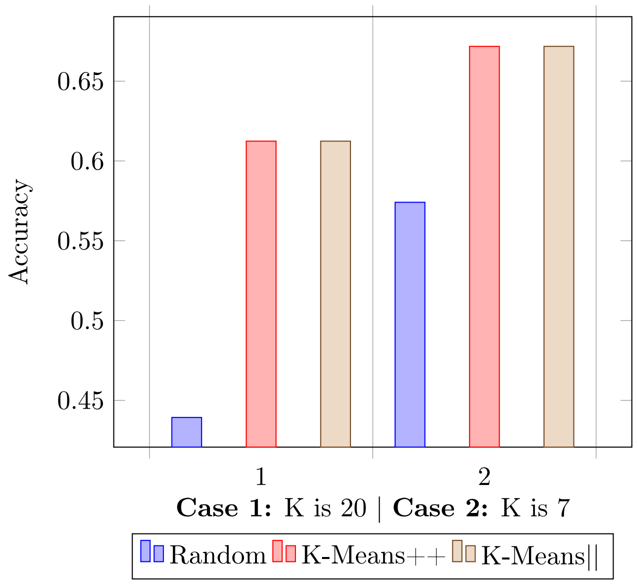
type of initialization	accuracy for K=20	total time
random	0.4394	84.6117
K-Means++	0.6124	156.9197
K-Means||	0.6124	90.6363

One can clearly see the advantage of a K-Means|| initialization relatively to a random initialization both in terms of speed and accuracy. Moreover, K-Means|| is faster than K-Means++ but achieve the same accuracy. Notice that when K-Means|| is running, the initializationsteps, which is the number of steps in K-Means|| is fixed to $r = 10$ and $l = K$. Now here are the results for $K = 20$:

type of initialization	accuracy for K=7	total time
random	0.5741	57.9025
K-Means++	0.6718	74.8596
K-Means||	0.6718	60.7070

To the sake of comparison between the two cases, the chart below summarize the information:

It can be noticed, that in this case, *random* is faster than K-Means|| arguably because the Initialisation-Steps parameter is fixed to 10 for *K-Means||* and then for a small K random is faster. But the difference in terms of accuracy is obious: K-Means|| beats *random* again. Moreover, as a kind of benchmark, a supervised learning classifier was implemented to try to establish how well the PSK algorithm was behaving. Using the same parsing, we ran a NaiveBayes Classifier. With the same parsing, the recovery is 0.4515 and one can improve the parsing to achieve around 0.5 of accuracy. Now we can study the runtime as the number of data points increase. We will sample uniformly in our data a percentage of data points and we will run the PSK algorithms, and Parallel K-means, with random initialization and K-Means|| initialization.



Now, in order to see how the 3 types of initializations are behaving with respect to the size N of the data, we sample from the original data set a percentage of points, uniformly, to have the same proportion of points in each cluster. The results are shown in Figure 2:

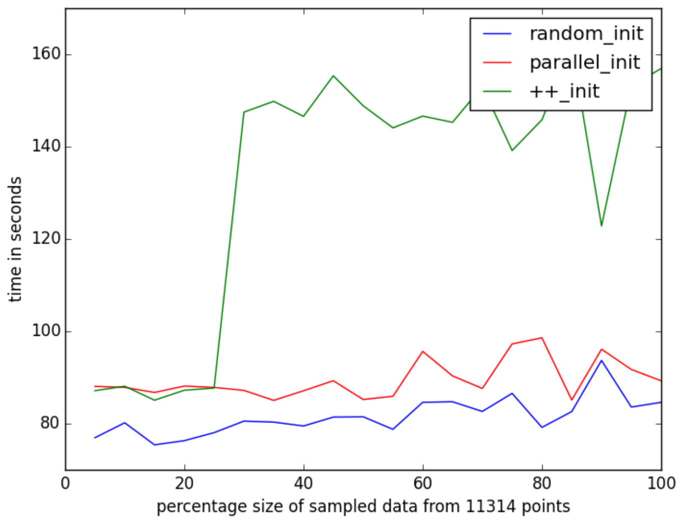


Figure 2: Runtime of Initializations with respect to Data size

7 Conclusion

While the scalability of the parallel procedure is improved, other ways of tackling the problem could be investigated. To mention some possible future developments of this project, it could be interesting to use the sherical norm instead of the euclidian distance: indeed, as the number of dimensions increase, the distance between any two points in the dataset is converging. This is of first concern when we are using the euclidian norm.
Another point is that, as Yumi Kondo puts it in [7], SK algorithm, and so PSK Algorithm by extansion are affected by a even few outliers. Implementing a parallel robustification of the PSK algorithm would be an interesting next step.

8 References

[1] Bahmani, B., Moseley, B., Vattani, A., Kumar, R., and Vassilvitskii, S. (2012). Scalable k-means++. Proceedings of the VLDB Endowment, 5(7), 622-633.

[2] Witten, D. M., and Tibshirani, R. (2012). A framework for feature selection in clustering. Journal of the American Statistical Association.

[3] Zhao, W., Ma, H., and He, Q. (2009). Parallel k-means clustering based on mapreduce. In Cloud computing (pp. 674-679). Springer Berlin Heidelberg.

[4] Bekkerman, R., Bilenko, M., and Langford, J. (Eds.). (2011). Scaling up machine learning: Parallel and distributed approaches. Cambridge University Press.

[5] Reed, J. W., Jiao, Y., Potok, T. E., Klump, B. A., Elmore, M. T., and Hurson, A. R. (2006, December). TF-ICF: A new term weighting scheme for clustering dynamic data streams. In Machine Learning and Applications, 2006. ICMLA'06. 5th International Conference on (pp. 258-263). IEEE.

[6] Liang, Y., Balcan, M. F., and Kanchanapally, V. (2013). Distributed pca and k-means clustering. In The Big Learning Workshop at NIPS (Vol. 2013).

[7] Kondo, Y. (2011). Robustification of the sparse K-means clustering algorithm (Doctoral dissertation, The University Of British Columbia (Vancouver).

[8] Boyd, S., and Vandenberghe, L. (2004). Convex optimization. Cambridge university press.