Data Mining Homework 3

Alessandro Monteleone 1883922

Academic Year: 2023/2024

Contents

1 Problem 1

1.1 K-means in 1 dimension

To solve this problem I propose this algorithm:

Algorithm 1 K-means 1-d

```
Require: $numClusters \leq points.size$
  procedure GET_K-MEANS_1-D($numClusters, points$)
    if $numClusters = points.size$ then
      return $points$
    end if
    SORT(points)
    $clusters \leftarrow$ K-MEANS 1-D($numClusters, points[: numClusters]$)
    SETUP_DATA_STRUCTURE(clusters)
    for point in points[numClusters:] do
      ADD(point,clusters[k])
      $lowering\_cost \leftarrow True$
      $i \leftarrow numClusters - 1$
      $last\_increased\_size\_cluster \leftarrow numClusters - 1$
      while lowering_cost and $i \geq 1$ do
        $Rcluster \leftarrow clusters[i]$
        $Lcluster \leftarrow clusters[i-1]$
        $lowering\_cost \leftarrow$ IS_LOWERING($Lcluster, Rcluster$)
        if lowering_cost then
          $last\_increased\_size\_cluster \leftarrow i - 1$
          $leftmost\_point \leftarrow$ POP_LEFTMOST($Rcluster$)
          ADD(leftmost_point ,Lcluster)
        else if $last\_increased\_size\_cluster = i - 1$ then
          $i \leftarrow i - 1$
          $lowering\_cost \leftarrow True$
        end if
      end while
    end for
    return GET_LIST_MEANS(clusters)
  end procedure
```

Now I will explain it and then I will prove its correctness together with its cost.

1.1.1 Algorithm explaination

The algorithm requires a number of clusters less than or equal than the number of points then it works in this way:

- If the number of clusters is equal to the number of points it place the means in the positions of the points (1 mean correspond to 1 point)
- Otherwise it starts by sorting the list of points and placing the k-means onto the first k poinst making a recursive call to itself with same number of clusters and the list made by the first k points
- The returned list is modified in order to become a data structure that is a list of clusters where the clusters are represented by the tuple (actual mean, list of points included in the cluster)
- Then it starts iterating on the remaining points (the first num_means points are removed)
- Since the list is ordered and we are visiting that from left to right at each iteration we add the new point to the last cluster (the one that includes the rightmost point) and we do it through the function ADD which append the point to the list of points in the cluster and recompute its mean
- Now we set up some variables that will be useful later which are:
  - lowering_cost: that is the variable that will control the next while loop
  - i: which is used to iterate on clusters

    - last_increased_size_cluster: which is needed in order to control which was the last cluster that has increased its size

- Then we start a while loop in which we modify our clusters until our costs stop decreasing or we have iterated on all clusters
- We use variables Rcluster (right cluster) and Lcluster (left cluster) to iterate on clusters
- Then we verify if moving the leftmost point in the Rcluster to the Lcluster lowers the sum of the costs of the two clusters
- If the cost is lowered the variable last_increased_size_cluster is updated to the index of the Lcluster and the leftmost point in the Rcluster is moved to the Lcluster (both means are then recomputed)
- Else If the cost is not lowered but in a previous step we have moved some point from Rcluster to Lcluster then we can continue iterating on the clusters so we update the variable i and we make the variable lowering_cost again true to continue iterating
- Upon the end of both the loops it is called a function that from our data structure it retrieves the list of k means and the result is returned in output

1.1.2 Proof of correctness and cost

Before to prove the correctness let's prove that the algorithm terminates, the proof is simple, in fact we can see that there are 2 loops: the for loop terminates since the list of points is finite, the while loop terminates in 2 cases:

- The cost is not lowering anymore: it must happen after at most n iterations on the same cluster where n is the number of points in that cluster
- We have iterated on all the clusters

So we have proven that the algorithm terminates, now let's show that it is correct.
When the number of cluster is equal to the number of points the prove of correctness is trivial, so we will focus on the case in which number of cluster is lower than the number of points.
Let's prove by induction that at the beginning of the j^{th} of the for loop our data structure contains the optimal solution for k cluster considering the first j-1 points.
Assume to start from j = number of clusters since we have placed the means before to start the loop, then at iteration j = number of clusters + 1 we surely have the optimal solution for the first j-1 points.
Now let's show that during the j^{th} iteration we build the optimal solution for the first j points.
The new point we are considering for the iteration is the rightmost points among the ones that we have considered until now (that's because the list of points is sorted) so it has to be included into the rightmost cluster for sure since there can't be another cluster nearest to that. Upon the addition to the new point to the cluster the new mean is computed and the cost is increased. Since the mean of the rightmost cluster has changed then the leftmost points of that cluster could be included in the cluster that is placed at the left side of the rightmost cluster so we check if moving some points to one cluster to the other decrease the sum of the cost of the two clusters and therefore the total cost of the clusters. When the cost stops decreasing then if the cluster to the left has added some points to itself then the position of its mean has changed and therefore it could be advantageous to give some of its leftmost points to the cluster at its left side, so the process continues until it arrives to the point in which either the cluster at the left doesn't accept any point so its mean remains the same and it is no sense to continue the iterations or the left cluster is the leftmost and it can't accept any more points (since it wouldn't lower the cost).
During these iterations we have decreased the total cost as much as possible since it is trivial that when we add a point which is the rightmost among the points considered until now the means of the clusters can move only to the right so we have only to check if it is useful to include points at the right and it wouldn't be necessary to try include points to the left since the cost of adding them will increase the total cost and that follows to the inductive hypothesis, in fact in the previous step, when each mean could be only in same position or at the left to the corresponding mean during this step, it was not advantageous to add points at left to the clusters so now that the means are moving to the right it is still so.
So at the end of the while loop we have a solution in which the cost is minimum and therefore that is the optimal solution for the first j points.
Now let's talk about the cost, let's consider that all the procedures have constant cost (then we will show that), we have a for loop that iterates for n times where n is the number of points, then the while loop iterates on clusters and for each cluster it moves its points until the cost stops decreasing and it can happen after at most O(n) iterations. So let's call k the number of clusters then the cost of the algorithm is O(n^2k).
In order to make sure that this is the cost let's analyze the costs of the procedures included in the algorithm:

- SETUP_DATA_STRUCTURE: this procedure simply take the k points and build a list of tuples point-[point] (since the clusters are constituted of 1 point the mean is in that point). Building this structure takes at most O(k) and since it is out of the loops and it is dominated by the cost that we have computed previously it doesn't matter in the total cost of the algorithm
- ADD: adds a point to a cluster, it is composed of 2 operations which are append the point to the list of the cluster and recompute the mean. Both this operations can be done in constant time, append is trivial, the computing of mean follows the formula:

  $$\mu_{new} = \frac{\mu_{old} \cdot n_c + newPoint}{n_c + 1} \tag{1}$$

  Where μ is the mean of the cluster (old or new), n_c is the number of points in the cluster before adding the new point and newPoint is the coordinate of the new point
- IS_LOWERING: this function simulate the removing of the leftmost point of a cluster and the addition of that into the other cluster (ADD function), compute the new cost and compare it with the old one . The ADD function we already know that can be executed in constant time, so let's introduce the removing of the point from a cluster (POP_LEFTMOST function), this function remove the leftmost point (it can be done in constant time, trivial) and then recompute the mean with a calculus similar to the one showed in (1) which is also constant.
- POP_LEFTMOST: we already introduced it in the previous part
- GET_LIST_MEANS: this simply extract the k means from the data structure containing the clusters. The cost is therefore O(k) but since it is out of the loops and its cost is dominated by the cost of the loops the total cost of the algorithm remains the same.

1.2 k-means 1-d approximation

The solution that I have found does the following: if I want to get k means and k is even then I take the farthest point at the right of the point 0 and I divide the space of the distance into k slices numerated from 1 to k, then I place k/2 means in positions $1 + 2a$ where a goes from 0 to k/2 - 1, then I place the remaining means in the same way with respect to the farthest point at the left of the point 0. If k is odd I simply place 1 mean in the point 0 and then I apply the method for k even for the remaining k-1 means.
Using this approach the formula to find the new cost will be the following:

$$NewCost = \sum_{i=1}^{k} \sum_{x \in \mu_i^P} (x - \mu_i)^2 \tag{2}$$

We use μ_i^P representing all the points x covered by the mean μ_i.
Now we can notice that $x - \mu_i$ can be at most as big as the radius covered by the mean and this is exactly $\frac{x_{l+}}{k}$ for positive points and $\frac{|x_{l-}|}{k}$ for negative points (x_{l+} is the rightmost point and x_{l-} is the leftmost point)
Now we can substitute the computed values inside the formula that becomes:

$$NewCost = \sum_{x \in P, x>0} \left(\frac{x_{l+}}{k}\right)^2 + \sum_{x \in P, x<0} \left(\frac{|x_{l-}|}{k}\right)^2 \tag{3}$$

In order to simplify again the formula we can say that this result is surely smaller or equal than if we choose the bigger between x_{l+} and $|x_{l-}|$ (let's call it x_l) so we haven't to treat differently negative and positive values anymore and the new formula will be:

$$\sum_{x \in P, x>0} \left(\frac{x_{l+}}{k}\right)^2 + \sum_{x \in P, x<0} \left(\frac{|x_{l-}|}{k}\right)^2 \leq \sum_{x \in P} \left(\frac{x_l}{k}\right)^2 \leq \frac{1}{k} \sum_{x \in P} x_l^2 \tag{4}$$

Given this formula I will choose $\frac{1}{k}$ as ϵ, anyway we can not state, as it is required from the problem, that in every case the new value is ϵ times the cost with one mean but for a special case this is true, in fact if all the points are grouped near the farthest points to the left and to the right then the cost for 1 mean will be almost $\sum_{x \in P} x_l^2$ so in this case we have showed that we can place the k-means in this way to achieve in worst case ϵ times the cost of the 1-means solution.

2 Problem 2

2.1 Code explaination

For this problem the code is contained in the file k_means_analysis.py.
Before answering the questions of this exercise I will explain what I have done in the code to do the experiments.
I have used the library sklearn which implements the functions for PCA and k-means++ and then I have created a class (Report) which handles the execution of the experiments and the storing of the results. The execution of the experiment is easy: first execute k-means++ and then runs PCA followed by k-means++. I have chosen to make the k-means algorithm stop either if it does 10 iterations or if the k-means cost difference between 2 subsequent iterations is less than 0.002, for the PCA instead I have chosen to keep $\frac{k}{\epsilon}$, with $\epsilon = 0.85$, components . For what concern the results I have stored the running times of the algorithms and the costs of the clusters, but I needed also to see if the produced clusters were similar to the expected ones. In order to do this I have compared each produced cluster with each expected cluster computing the Jaccard similarity between them, so for each produced cluster I have taken the max Jaccard similarity among all the similarities computed for that cluster then I have summed all the results of the clusters and I have divided the total for the number of clusters. In this way I have obtained the mean of the similarities of the clusters represented as a number between 1 (the clusters are the same as the expected ones) and 0 (the clusters are completely different from the expected ones).
It was difficult to run the algorithm since the big matrices to generate so I have decided to make the float precision a parameter that can be set (in order to maximize the number of reproducible cases I have chosen 16-bit floats).
I saved the results in a file called "experiment_results.json" but I have also a function to visualize them in a graph.
As last I have set a main function to execute different tasks:

- do experiment: it requires to specify "do experiment" as first input argument. It executes the experiments and store the result in the file "experiment_results.json" (this can be changed modifying the constant REPORT_FILE in the python file)
- combine results: it requires to specify "combine results" as first input argument. It is possible that you have to execute the experiments different times because if the matrices are too big the program can crash, if it is the case this function is used to merge the results stored in 2 files (the results of second files are put in the first), you obviously have to specify the 2 files to merge as input arguments
- reformat time: it requires to specify "reformat time" as first input argument. It may be good to have the running time expressed in minutes, seconds and milliseconds instead of milliseconds only, this function does this, you have to specify only the file on which apply the new formatting
- print metric: it requires to specify "print metric" as first input argument. It prints some metrics in a chart, it requires as input the name of the file from which to load the results , the metrics to plot separated by comma (they should be metrics that have sense to plot together), the names to assign to each metric, the name to assign to y axis.

2.2 Analysis of Jaccard similarity for k-means only

Now let's look at the results to see how the changes in parameters affect the Jaccard similarity:

- n: the increasing of this parameter seems not to affect the similarity
- k: the increasing of this parameter seems not to affect the similarity
- d: this parameter do affect the Jaccard similarity because increasing it results in adding more noise, still this won't affect the similarity too much if d is not increased a lot
- σ^2: this parameter strongly affect the similarity in fact increasing it results in making the noise stronger, when it goes to 0.5 as we can see in fig. (1) the similarity completely fall down

2.3 Comparison between k-means only and PCA + k-means in terms of Jaccard similarity

For Jaccard similarity the difference between k-means only and PCA + k-means can be seen in some behaviours of PCA + k-means, we can see them in fig (1), what we can notice is that PCA seems to help with increasing n and k but it seems to suffer a lot from the increasing of d and σ^2. This makes sense since with more points or more clusters and small noise it could help because the components with more variance could be the ones of the I_k

matrices, with more noise (increasing d or σ^2) instead the components given in output by PCA will be the ones of the matrices $N^{k,d}$

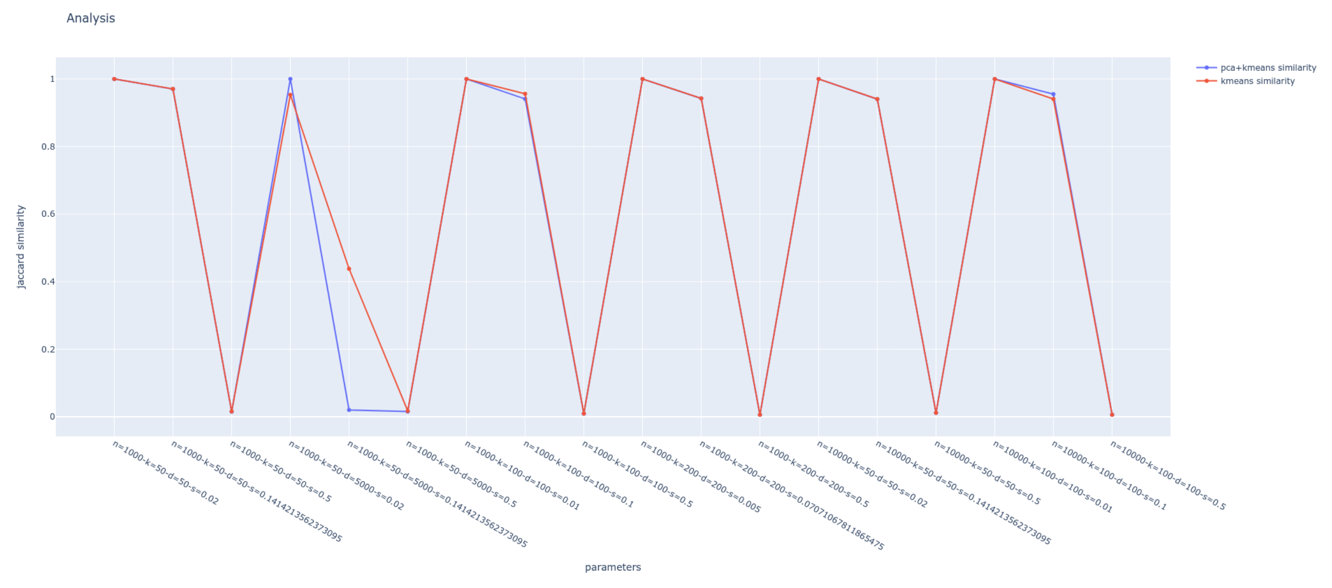

Figure 1: Comparison between k-means++ alone and PCA + k-means++

2.4 Running time analysis and comparison between k-means only and PCA + k-means

For this metric the increasing of all the parameters will increase the running time, in particular we can see that the increasing of d in particular increase a lot the running time.
Comparing k-means only and PCA + k-means we can notice that all the increasing in the parameters will still result in an increasing of the running time but it will be smaller and in some cases PCA + k-means will be way way better in terms of running time. We can see these result in fig (2), notice that the running time plotted for PCA + k-means is the running time of both the algorithms not the one of PCA only and apart from the first cases (with smaller parameters) the cost of PCA + k-means will be better than k-means only

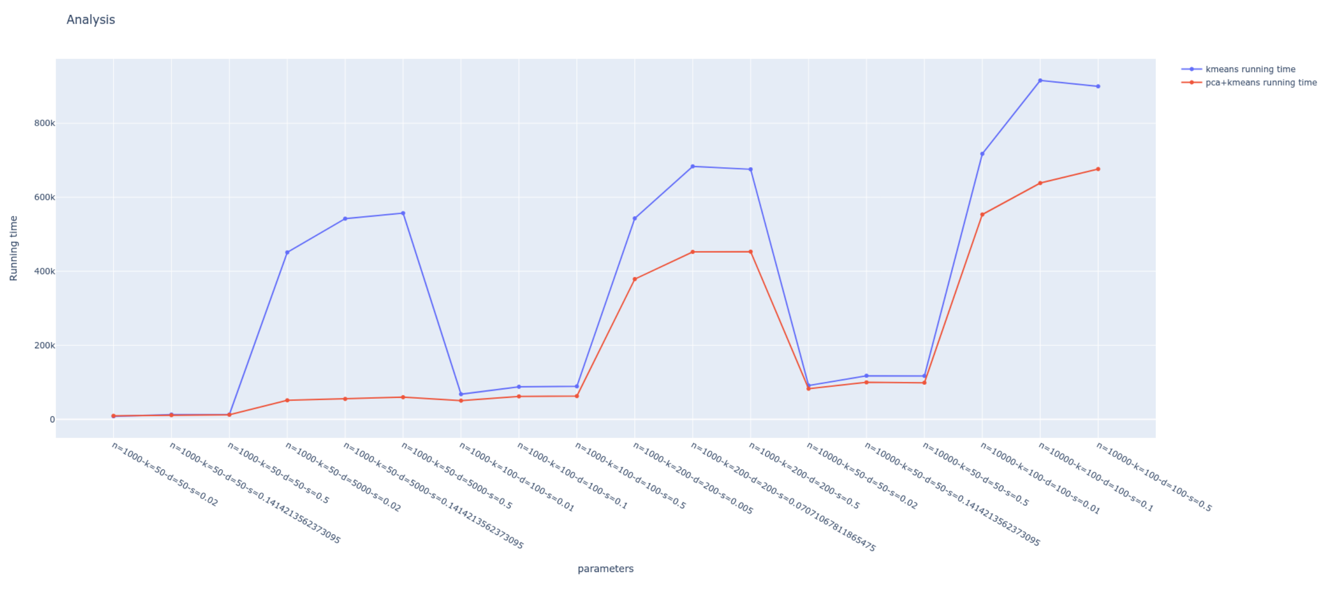

Figure 2: Comparison between k-means++ alone and PCA + k-means++

3 Problem 3

3.1 Code explaination

For this problem the code is contained in files: k_means_analysis_amazon_products.py, utils.py, app.py. The clustering algorithm that I have chosen is k-means++, to apply it to the data I have built 2 functions: process_raw_data and apply_feature_engineering, both of them can be used to run k-means either on a certain number of clusters or on all the number of clusters from 1 to max_means (this is an argument given in input to the functions). Once the k-means algorithm is executed the functions can return variances, running times or the list of clustered records. Now for each function I will explain what it does.

3.1.1 process_raw_data

This function is used to process the raw data in order to give them to the k-means algorithm, it is required since k-means accept numerical vectors and not strings. However I tried to reduce the preprocessing as much as possible in order to let the data be raw. What I have done is I took the data and for each record I split it into words using the library nltk and then I converted each word into a number. For this last thing I built a dictionary containing for each word its id (given incrementally), then since k-means requires that all the vectors must have the same dimensions I simply added all 0 padding to the vectors.

3.1.2 apply_feature_engineering

This function is used to preprocess the data and apply feature engineering before to use k-means algorithm, now I will only explain what the code does then in next section I will explain why certain preprocessing have been done. This time the data are loaded with pandas and the preprocessing is applied only to the description of the products. There are various techniques that can be applied during preprocessing, some of them are exclusive (you have to choose 1 among them) some others can be always applied, the former ones are the following:

- TFIDF: if this technique is selected the data are preprocessed and each word is substituted with its TFIDF
- Minwise hashing: when this technique is selected each record is substituted with its representation produced through minwise hashing , the code is exactly the same as in Homework 2
- One hot encoding: if this technique is selected then let's return to the dictionary built for raw data, well this is used to place the number 1 in an all 0 vector in the position corresponding to the id of the word if the word is present in the record

The other techniques that can be always applied are: normalization, centralize and PCA. Let's spend few words on each:

- Normalization: once the matrix containing all the preprocessed records is produced it is divided by its norma
- Centering: a central vector is produced as mean between the vectors in the matrix, then it is subtracted to each vector of the matrix
- PCA: once the matrix containing all the preprocessed records is produced it is applied the function PCA of the library sklearn using the number of components specified in input

3.1.3 main function

The main function of the file k_means_analysis_amazon_products.py it is used to plot the data, but before to do this you have to specify what you want to plot, let's explain the required input arguments briefly:

- argument1: you have to specify what metrics to plot, they are "elbow curve" or "running times", you can also specify both ("elbow curve and running times") in that case they will be plotted in 2 charts but in the same page
- argument2: you have to specify a comma separated list of techniques of which you are interested in seeing the metrics, example "tfidf,minwise hashing,raw". If you want you can also specify addictional preprocessing to add such as PCA or normalization, example "tfidf pca,minwise hashing normalized,raw"
- argument3: this argument is not mandatory, it is needed if you want to use additional fields other than the "description" field during clustering, the list of fields must be a comma separated list, example: "price, prime"
- argument4: this argument is not mandatory, it represent the maximum number of means for which you want to run the techniques, if not specified the constant MAX_MEANS will be used

3.1.4 app.py

This file is a simple flask application, it is used to visualize the clusters produced applying the various techniques. It starts from the main page which contains a form in which you can specify which technique you want to use to preprocess data, how many clusters to use to run k-means and which cluster to visualize, remember number of custers starts from 0 so if you have specified to use 4 clusters to do k-means the clusters that you cold visualize will be either 0 or 1 or 2 or 3.
The result of clustering will be displayed in a table containing for each product all its fields, then if you want to experiment another technique or visualize a different cluster you have a button at the top left of the page to return to the form page.

3.2 Report

For this experiment I have used k-means++ algorithm, I won't spent too much time explaining it since we have seen it during the lectures, anyway it is an algorithm that at the beginning places the k means randomly following a probability distribution based on the distance of the nearest mean already placed (a mean is placed on a point with a probability proportional to its distance from the nearest mean squared), then the positions of the means is adjusted in various steps reducing the total cost each time.
Now let's discuss the results.
In order to do produce better results than the ones of raw data I have chosen to remove the url field since it is different for each record so it doesn't help in clustering, I have also tried to remove other fields and leave only the description field to perform clustering only on that but it has made the clusters worse and the elbow curve also present a worsening as can be seen in fig (3).
Another thing that I have done to help the clustering of the "description" field is that I have tried to delete some stopwords and punctuation in order to make the descriptions more clear hoping to leave only valuable words in terms of clustering.
As last thing I have also removed the duplicates in the data since they were useless in clustering and gave more weight to some records without a real motivation.
Then I ran k-means with different techniques to see which was the best, now I will explain the results obtained for each technique and the comparison with the running of k-means using raw data, all the elbow curves and running times can be seen respectively in figures (4) and (5).

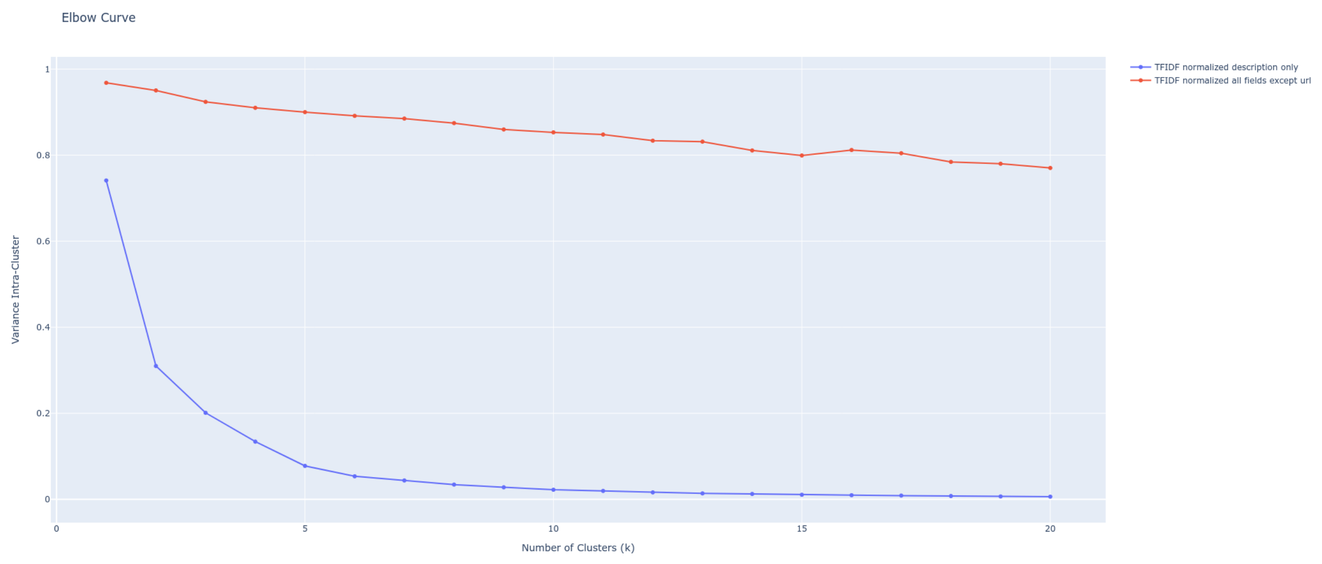

Figure 3: Comparison between the elbow curves produced using TFIDF with only description field and with all the fields except for url

3.2.1 Minwise Hashing

One technique that came into my mind for representing strings as numbers was minwise hashing, so I have tried to use it. The results unfortunately were really bad, it didn't help in clustering, the running time was very good but the clusters themselves weren't good, some clusters had very few records and others had lots of them and the elbow curve also, although it was better than the one of raw data, was bad. I think that this is because with this technique we could get hashes of few words among all the ones in the description and since we are using hash functions we could get really distant values in the different dimensions.

3.2.2 One Hot Encoding

This technique differently from the previous one provides more assurances in fact the values in the dimensions of the vector representing each record can be either 1 or 0 and all the words are represented. This representation was way way better than the previous, the elbow curve was more clear and the clusters finally made sense, looking at them I was able to find some common features. The running time, although it was worse than the one of minwise hashing, was still way better than the one of raw data.

3.2.3 TFIDF

The last technique I have tried was TFIDF it is similar to the previous but it tries to represent better the frequency of the words in the records. This technique produced an elbow curve equal to the one of one hot encoding (it makes sense since the way they have to represent the data are similar), however the produced clusters were not as good as the ones of the other technique, they were still good and way better than minwise hashing but it was not clear for some clusters why certain records were there and then sometimes there were clusters with few elements. I think this could be duded to the fact that terms that appears in lots of records are penalized, increasing the frequency of a term in all the records will make its TFIDF value go to 0 making records without that element more similar to the ones which contains it and it could be bad in clustering, for example if I have a lot of records with that term and one that doesn't contain it I may want it to be in a different cluster since it is so different from all the others. Another think that is important to notice is that with TFIDF I will assign higher values to the words that appears in few records, so differently from the case of one hot encodings the records without certain words will be farther to the records containing those. For what concern the running time it is better than the one of one hot encodings.

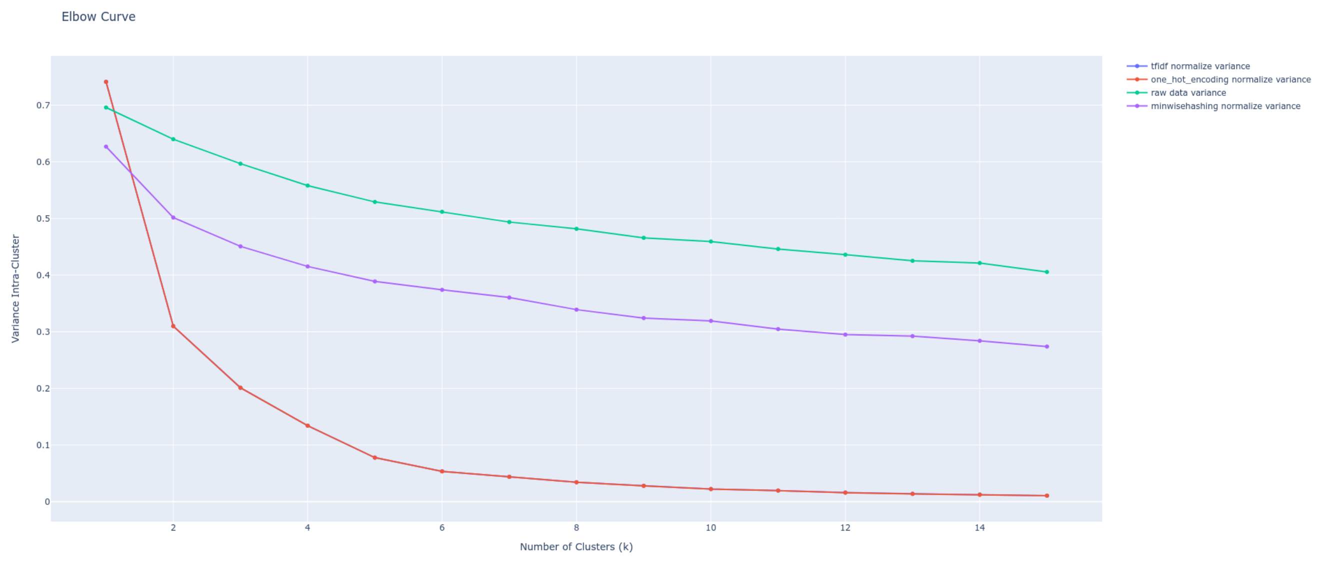

Figure 4: Comparison between the elbow curves produced using different techniques (TFIDF and one hot encoding are overlapping)

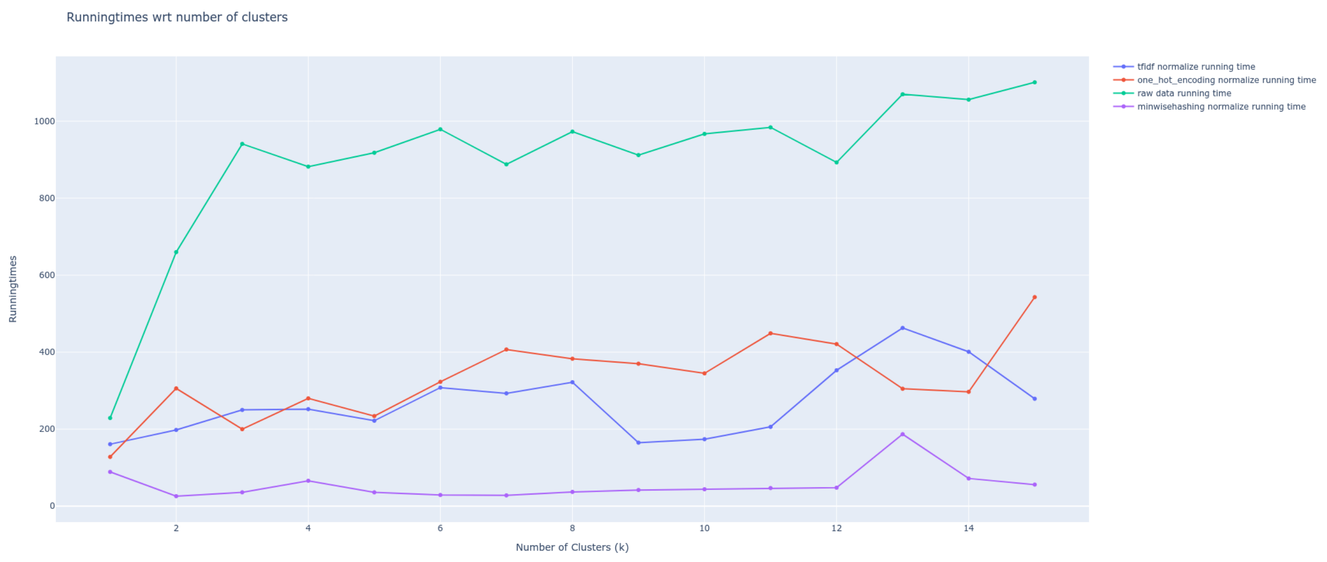

Figure 5: Comparison between the running times of different techniques

3.2.4 Analysis of best technique

I think that the best technique among the ones previously showed is one hot encoding since it has a good elbow curve and differently from TFIDF seems to have clusters that make more sense. To say this what I have done is I have choose to plot 5 clusters for each technique and I have tried to analyze them. For each cluster I tried to assign it some features which I found common in the contained records, it was really difficult for some techniques but with one hot encoding it was way better.
In addition to this techniques I have also used other manipulations on the result data, now I will present them and their effects on the clustering, I will use TFIDF as the reference technique since it has both running times and clusters that can be improved.

3.2.5 Normalization and Centering

2 useful manipulations to apply to the data are normalization and clustering. I tried both and what I have noticed is that both of them helps in reducing the running time in particular in the cases with few means, we can see this in figure (6). It is not clear which manipulation reduce more the running times since it changes depending on the number of means and it can change running the algorithm different times, then the differences are not so important in fact we are talking about order of magnitude of milliseconds. For which concern the clustering neither combination of these seems to affect the clusters in a valuable way, they are more or less as meaningful as without the manipulations. One thing that I have found to be important of normalization is that is really useful to represent together the elbow curves of the various techniques, in fact they could differ a lot and plotting them "raw" would give informations only on the ones with the highest values.

3.2.6 PCA

This manipulation was game changing, I have experimented with it and I found out that not only the running times decreased a lot but also the clusters were better. Since PCA keeps only the most valuable features in terms of variance I was able to reduce the dimensions of the vectors representing the records and also to produce better clusters, I have found out that even with really few features (3 or 4) the clusters were meaningful and the running times dropped down, we can see it in figure (7). Looking at the new clusters I could even say that they were better than the ones produced by one hot encoding. Although this manipulation made all better the elbow curve remained the same (and so was also for normalization and centering).

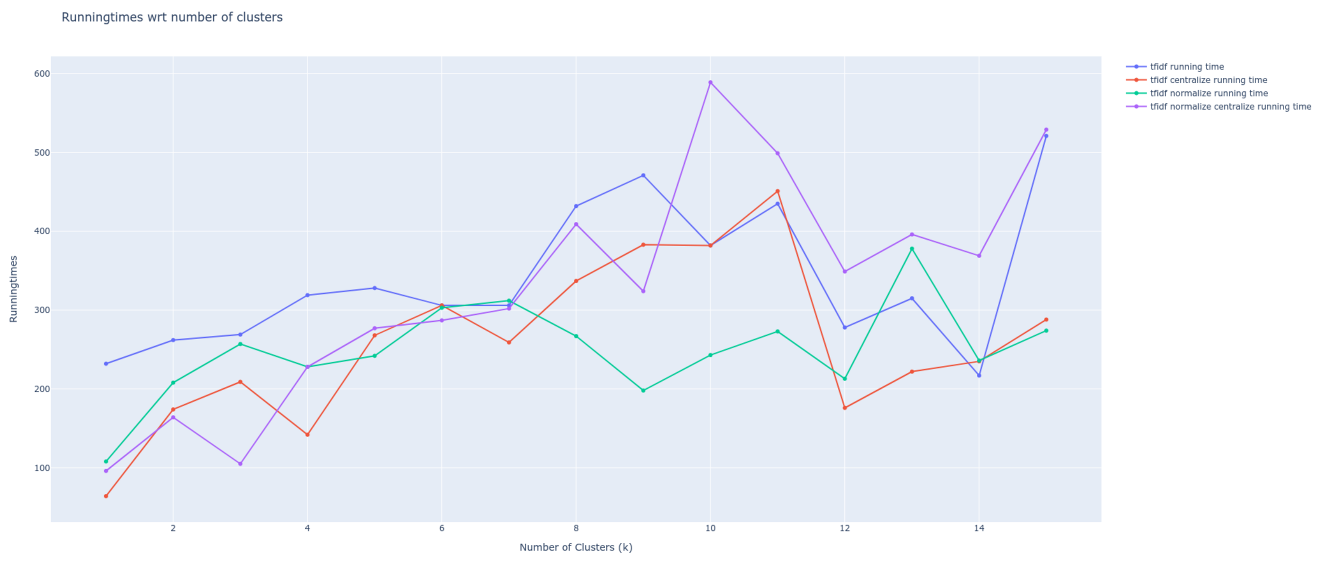

Figure 6: Comparison between the running times of clustering done through TFIDF plus combinations of centering and normalization

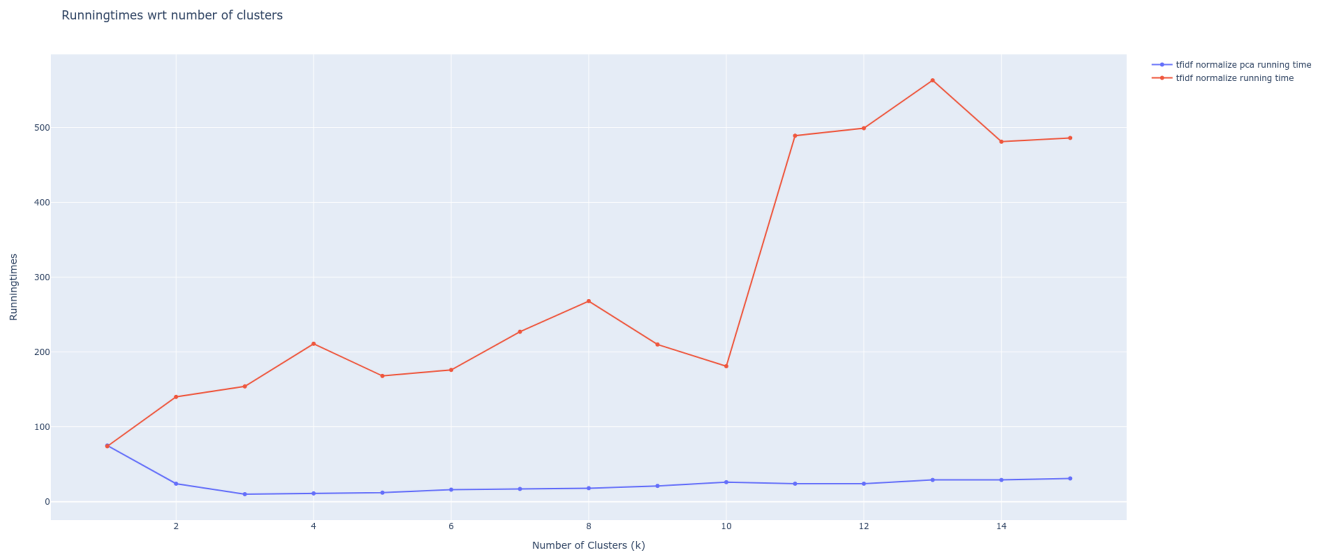

Figure 7: Comparison between the running times of clustering done through TFIDF within and without PCA

3.3 Conclusions

To conclude what I have found out is that feature engineering is very useful for clustering these data, one hot encodings technique seems to fit better to represent the "description" field and PCA is very useful to reduce running time and produce better clusters. Anyway I think that the description field could be represented better, in fact as shown in fig (3) if this is the only field used the elbow curve is way way worse. To fix this thing it has to be produced a vector representing better the argument of the description, what I think is to use something similar to word embeddings that may be done exploiting machine learning techniques. By the way using the techniques we have studied until now it seems to produce good results probably because some of the other fields reflect the things which are in the descriptions, for example the cables for gpus have lower prices.