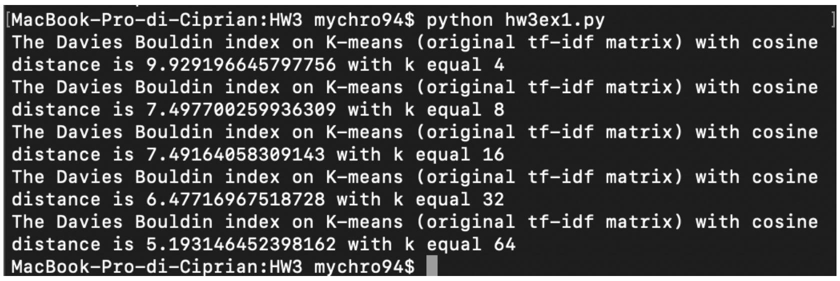
Exercise 1

For this exercise the file of job announcements created in the previous homework will be used, that contains already the processed announcements. Each job is written on a line. For clustering all the documents also the inverted index, generated in homework 2, was used. In particular, the first step done was to create the *tf-idf* values matrix m. Its shape will be 4259x18929, since the announcements are 4259 and the number of words is 18929. Therefore, in m, at position matrix[i][j] there will be the *tf-idf* value of word j in document i. The value is computed as in homework 2, it is the product between the *term frequency* and the idf, that is the log $\frac{N}{docF}$, where $docF$ is the document frequency.

The K-means algorithm was applied on the matrix m, with the *cosine distance* metric. The used library is *nltk.cluster.kmeans.* The value for k was found doing many trials, for example invoking the k-means algorithm with $k = 2, 4, 8, 16...$, and observing that the values according to the *Davies-Bouldin* index decrease.

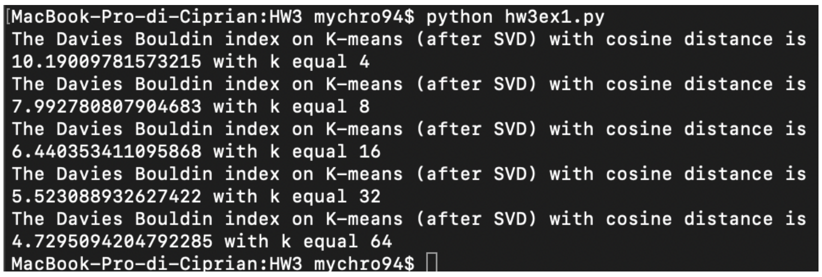


```
[MacBook-Pro-di-Ciprian:HW3 mychro94$ python hw3ex1.py                      ]
The Davies Bouldin index on K-means (original tf-idf matrix) with cosine
distance is 9.929196645797756 with k equal 4
The Davies Bouldin index on K-means (original tf-idf matrix) with cosine
distance is 7.497700259936309 with k equal 8
The Davies Bouldin index on K-means (original tf-idf matrix) with cosine
distance is 7.49164058309143 with k equal 16
The Davies Bouldin index on K-means (original tf-idf matrix) with cosine
distance is 6.47716967518728 with k equal 32
The Davies Bouldin index on K-means (original tf-idf matrix) with cosine
distance is 5.193146452398162 with k equal 64
MacBook-Pro-di-Ciprian:HW3 mychro94$
```

As we can see, increasing k, the distance within clusters decreases. So, with a higher value of k there will be formed more precise and specific clusters of job announcements. As we will see at the end, also $k = 4$ will be a good value that clusters similar annoucements. The main reason for which k-means algorithm was chosen was because of its running time, that is $O(ndkT)$, where n is the number of d-dimensional vectors, k is the number of clusters and T is the number of iterations needed until convergence. We could have chosen other clustering approaches such as hierarchical bottom-up, but its cost is prohibitive.

Successively SVD has been applied, that leads to a low-dimensional representation of a high-dimensional matrix. The singular-value decomposition allows to maintain the informations in the high-dimensional matrix, and at the same time eliminates the less important parts of the matrix, in order to reduce its dimensions. Without exceeding into details, this approach computes three other matrices, U, S and Vh starting from the matrix m defined above. From *numpy*, the *svd* function was used to achieve the decomposition. The matrix S is a diagonal matrix, and the sum of the squares of the diagonal returns the total energy of the matrix m. We can reduce its size, by maintaining 90% of the energy just by keeping the first j singular values, of which sum of squares, is at least 90% of the total energy. Indeed, in the implementation, from 4259 singular values we can mantain only 1579 of them, that allows us to maintain the 90% of precision. After having reduced the number of singular values, we need to eliminate the correspondent colums in the matrix U and the correspondent rows in the matrix Vh. Then, we need to connect the documents to the strength of the words, that is represented by the singular values, therefore we multiply the matrices U and S.

Obtained the US matrix, we can again apply the k-means algorithm with the same parameters as before. We will observe an increasing speed of the running algorithm, because this matrix is much smaller than m. For the same values of k, we can see that the Davies-Bouldin index values are pretty much the same.



```
[MacBook-Pro-di-Ciprian:HW3 mychro94$ python hw3ex1.py
 The Davies Bouldin index on K-means (after SVD) with cosine distance is
 10.19009781573215 with k equal 4
 The Davies Bouldin index on K-means (after SVD) with cosine distance is
 7.992780807904683 with k equal 8
 The Davies Bouldin index on K-means (after SVD) with cosine distance is
 6.440353411095868 with k equal 16
 The Davies Bouldin index on K-means (after SVD) with cosine distance is
 5.523088932627422 with k equal 32
 The Davies Bouldin index on K-means (after SVD) with cosine distance is
 4.7295094204792285 with k equal 64
 MacBook-Pro-di-Ciprian:HW3 mychro94$ 
```

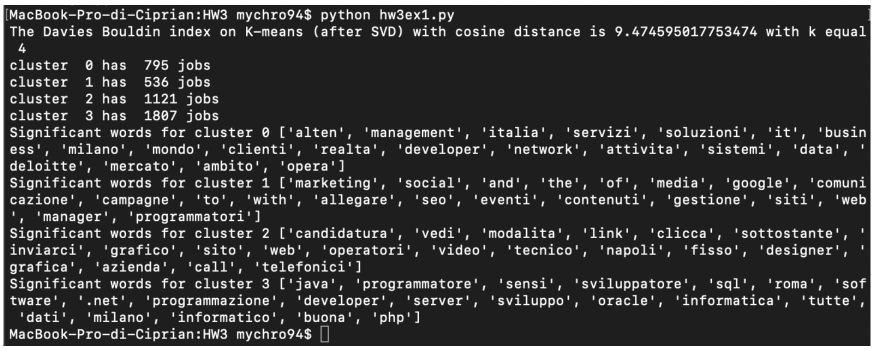
The differences observed are due to the fact that the initial means are random, so the final cluster centers can vary, therefore also the final clusters will be slightly different.

In the end, a good value of k can be 200, since the index value decreases by a very small value, increasing k. With this value of k we will have very specific clusters, while with a value of k equal to 3 or 4 we should have general clusters that include various job announcements, even though most of the announcements should contain more or less the same topics within a cluster.

Regarding the last point of the problem, the one of finding the most descriptive words of each cluster, the *tf-idf* matrix m was used. The following approach was used:

```
for each cluster c
    dictionary tfidf
    for each announcement in c
        for each word in m[announcement]
            update tfidf[word]
    print top 20 tfidf values from the dictionary
```

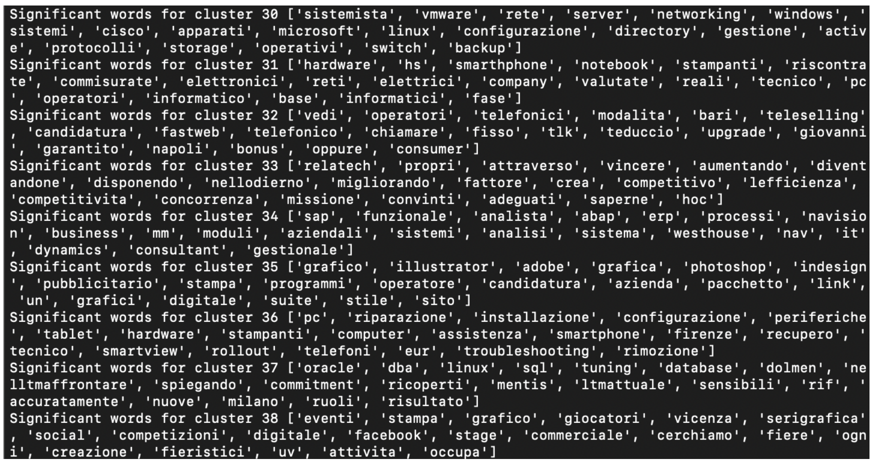
In practice, the approach was to sum all the tf-idf values of each word encountered in whatever document in the cluster. After being done with the cluster, the top 20 words describing the cluster according to the highest tfidf values were printed. This approach was used because it tends to give a higher importance to words that appear many times in the announcements, but with smaller document frequency. So, if the tfidf values were used for clustering the documents, then the most influencing words will have the highest value also, and it seems right to search for descriptive words in this manner.



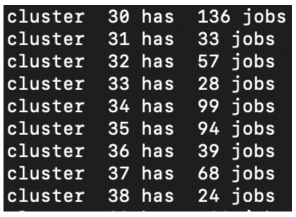
```
MacBook-Pro-di-Ciprian:HW3 mychro94$ python hw3ex1.py
The Davies Bouldin index on K-means (after SVD) with cosine distance is 9.474595017753474 with k equal
 4
cluster  0 has  795 jobs
cluster  1 has  536 jobs
cluster  2 has  1121 jobs
cluster  3 has  1807 jobs
Significant words for cluster 0 ['alten', 'management', 'italia', 'servizi', 'soluzioni', 'it', 'busin
ess', 'milano', 'mondo', 'clienti', 'realta', 'developer', 'network', 'attivita', 'sistemi', 'data', '
deloitte', 'mercato', 'ambito', 'opera']
Significant words for cluster 1 ['marketing', 'social', 'and', 'the', 'of', 'media', 'google', 'comuni
cazione', 'campagne', 'to', 'with', 'allegare', 'seo', 'eventi', 'contenuti', 'gestione', 'siti', 'web
', 'manager', 'programmatori']
Significant words for cluster 2 ['candidatura', 'vedi', 'modalita', 'link', 'clicca', 'sottostante', '
inviarci', 'grafico', 'sito', 'web', 'operatori', 'video', 'tecnico', 'napoli', 'fisso', 'designer', '
grafica', 'azienda', 'call', 'telefonici']
Significant words for cluster 3 ['java', 'programmatore', 'sensi', 'sviluppatore', 'sql', 'roma', 'sof
tware', '.net', 'programmazione', 'developer', 'server', 'sviluppo', 'oracle', 'informatica', 'tutte',
 'dati', 'milano', 'informatico', 'buona', 'php']
MacBook-Pro-di-Ciprian:HW3 mychro94$ 
```

As we can see, the 4-means was been run after having applied SVD, and the words have been retrieved using the matrix *m*. There can be observed the number of jobs each cluster has, and the top 20 most significant words for each cluster. Indeed, since the job announcements were gathered from the kijiji subsection *Informatica/Grafica/Web*, that ideally contains 3 categories, we can observe that the first cluster contains announcements about network, sistemi, developer or servizi, the second cluster contains words like programmatori, manager gestione, contenuti, siti, social. The third is containing video, grafico, sito, web, designer, and the last one java, programmatore, sviluppatore, sql, .net php or oracle. In conclusion it seems that the clusters are well defined, in particular the last one is about *informatica*, the third about *grafica*, the second about *web* and the first about business activities.

With a greater k we will expect that clusters contain much similar documents. Here are some examples of significant words in different clusters, after 64-means has been run. There can be observed distinct clusters, such as 31 related to hardware, or 35 related to grafica and photoshop.



```
Significant words for cluster 30 ['sistemista', 'vmware', 'rete', 'server', 'networking', 'windows', '
sistemi', 'cisco', 'apparati', 'microsoft', 'linux', 'configurazione', 'directory', 'gestione', 'activ
e', 'protocolli', 'storage', 'operativi', 'switch', 'backup']
Significant words for cluster 31 ['hardware', 'hs', 'smarthphone', 'notebook', 'stampanti', 'riscontra
te', 'commisurate', 'elettronici', 'reti', 'elettrici', 'company', 'valutate', 'reali', 'tecnico', 'pc
', 'operatori', 'informatico', 'base', 'informatici', 'fase']
Significant words for cluster 32 ['vedi', 'operatori', 'telefonici', 'modalita', 'bari', 'teleselling'
, 'candidatura', 'fastweb', 'telefonico', 'chiamare', 'fisso', 'tlk', 'teduccio', 'upgrade', 'giovanni
', 'garantito', 'napoli', 'bonus', 'oppure', 'consumer']
Significant words for cluster 33 ['relatech', 'propri', 'attraverso', 'vincere', 'aumentando', 'divent
andone', 'disponendo', 'nellodierno', 'migliorando', 'fattore', 'crea', 'competitivo', 'lefficienza',
'competitivita', 'concorrenza', 'missione', 'convinti', 'adeguati', 'saperne', 'hoc']
Significant words for cluster 34 ['sap', 'funzionale', 'analista', 'abap', 'erp', 'processi', 'navisio
n', 'business', 'mm', 'moduli', 'aziendali', 'sistemi', 'analisi', 'sistema', 'westhouse', 'nav', 'it'
, 'dynamics', 'consultant', 'gestionale']
Significant words for cluster 35 ['grafico', 'illustrator', 'adobe', 'grafica', 'photoshop', 'indesign
', 'pubblicitario', 'stampa', 'programmi', 'operatore', 'candidatura', 'azienda', 'pacchetto', 'link',
 'un', 'grafici', 'digitale', 'suite', 'stile', 'sito']
Significant words for cluster 36 ['pc', 'riparazione', 'installazione', 'configurazione', 'periferiche
', 'tablet', 'hardware', 'stampanti', 'computer', 'assistenza', 'smartphone', 'firenze', 'recupero', '
tecnico', 'smartview', 'rollout', 'telefoni', 'eur', 'troubleshooting', 'rimozione']
Significant words for cluster 37 ['oracle', 'dba', 'linux', 'sql', 'tuning', 'database', 'dolmen', 'ne
lltmaffrontare', 'spiegando', 'commitment', 'ricoperti', 'mentis', 'ltmattuale', 'sensibili', 'rif', '
accuratamente', 'nuove', 'milano', 'ruoli', 'risultato']
Significant words for cluster 38 ['eventi', 'stampa', 'grafico', 'giocatori', 'vicenza', 'serigrafica'
, 'social', 'competizioni', 'digitale', 'facebook', 'stage', 'commerciale', 'cerchiamo', 'fiere', 'ogn
i', 'creazione', 'fieristici', 'uv', 'attivita', 'occupa']
```



```
cluster  30 has  136 jobs
cluster  31 has  33 jobs
cluster  32 has  57 jobs
cluster  33 has  28 jobs
cluster  34 has  99 jobs
cluster  35 has  94 jobs
cluster  36 has  39 jobs
cluster  37 has  68 jobs
cluster  38 has  24 jobs
```

The python file can be executed from terminal giving *python hw3ex1.py*.

Exercise 2.1

For solving this problem we must follow a dynamic programming technique. With this approach we can use some data structure in order to save information about the solution of the same problem, but at previous steps. In fact, the solution provided will use a matrix *D*. Let's first define the k-means problem. Let $x_1, ..., x_n$ be an input array of n numbers sorted in non-descending order. The problem of 1-D k-means clustering is defined as assigning elements of the input 1-D array into k clusters so that the sum of squares of within-cluster distances from each element to its corresponding cluster mean is minimized. We refer to this sum as within-cluster sum of squares. We record the corresponding minimum sum of squares of within-cluster in entry D[i,m] of an n + 1 by k + 1 matrix D. Thus D[n, k] is the minimum sum of squares of within-cluster value to the original problem.
Let $CC(i,j) = \sum_{l=i}^{j}(x_l - u_{i,j})^2$ be the cost of grouping $x_i ... x_j$ points into one cluster with the optimal choice of centroid, $u_{i,j} = \dfrac{1}{j-i+1}\sum_{l=i}^{j} x_l$, that is the mean of the points.
In order to proceed we need to define a lemma.
Lemma 1. *It takes O(n) time and space to construct a data structure that computes CC(i,j) in constant time.*
Proof. This is a standard application of prefix sums. By definition:
$CC(i,j) = \sum_{l=i}^{j}(x_l - u_{i,j})^2 = \sum_{l=i}^{j} x_l^2 + u_{i,j}^2 - 2x_l u_{i,j} = (j-i+1)u_{i,j}^2 + u_{i,j}\sum_{l=i}^{j} x_l + \sum_{l=i}^{j} x_l^2$
Using prefix sum arrays of $x_1, ..., x_n$ and $x_1^2, ..., x_n^2$ computing the centroid $u_{i,j}$ and the sums takes O(1) time.
The algorithm finds the optimal clustering using i clusters for all prefixes of the points $x_1, ..., x_m$, for $m = 1, ..., n$, and $i = 1, ..., k$ with Dynamic Programming as follows. Let D[i][m] be the cost of optimally clustering $x_1, ..., x_m$ into i clusters. If $i = 1$ the cost of optimally clustering $x_1, ..., x_m$ is the cluster cost CC(1, m). That is, $D[1][m] = CC(1, m)$ for all m. By the lemma, this takes O(n) time.

For $i > 1$

$$D[i][m] = \min_{j=1}^{m} D[i-1][j-1] + CC(j,m) \tag{1}$$

Notice that $D[i-1][j-1]$ is the cost of optimally clustering $x_1, ..., x_{j-1}$ into $i-1$ clusters and $CC(j,m)$ is the cost of clustering $x_j, ..., x_m$ into one cluster. This makes x_j the first point in the last and rightmost cluster.

Proof of optimality. As mentioned above, if $i = 1$, $D[1][m] = CC(1, m)$, for any value of m. Now we assume that the equation (1) holds, so we will prove that also $D[1][m + 1]$ holds.

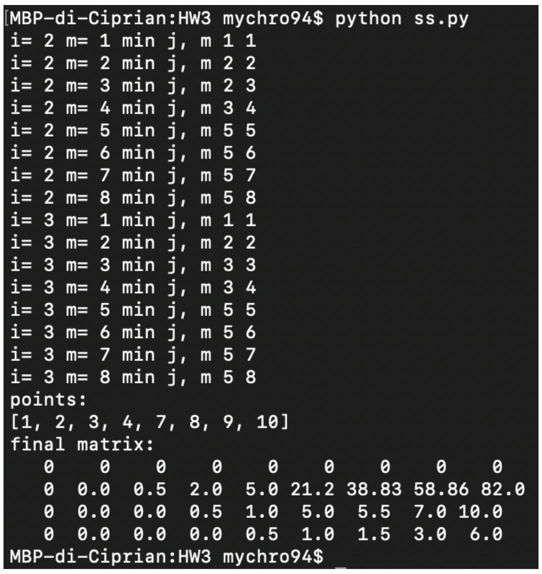
$$D[i][m + 1] = \min_{j=1}^{m+1} D[i - 1][j - 1] + CC(j, m + 1) \tag{2}$$

Using the inductive step

$$\begin{gathered} D[i][m + 1] = min(D[i][m], (D[i - 1][m] + CC(m, m + 1))) \\ = \min(\min_{j-1}^{m} D[i - 1][j - 1] + CC(j, m), (D[i - 1][m] + CC(m, m + 1))) \\ = \min_{j=1}^{m+1} D[i - 1][j - 1] + CC(j, m + 1) \end{gathered}$$

The time complexity of this algorithm is $O(kn^2)$

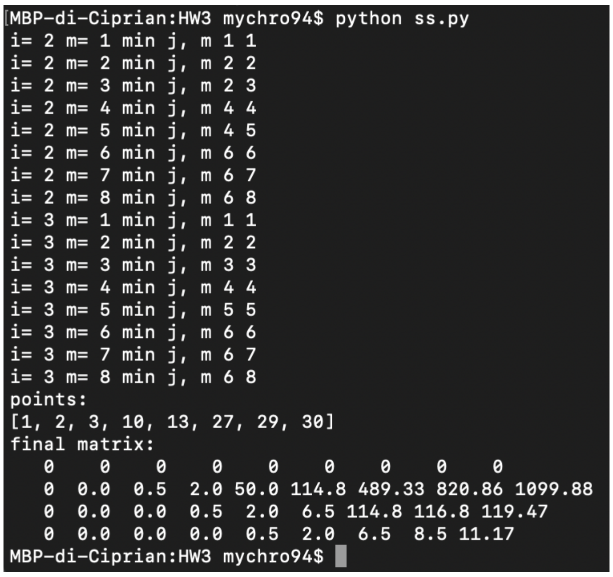
The Dynamic Programming algorithm can be sped up significantly to $O(kn)$ time by reducing the time to compute each row of D to O(n) time instead of $O(n^2)$ time.
The optimal solution can be found with backtracing after having run the dynamic programming algorithm. From the last cell of the matrix, that is given by the values of n and k, we can see which call generated the value stored in D[n][k], and so on until we get to a base case.
Follows the output of the implementation of the algorithm:



```
MBP-di-Ciprian:HW3 mychro94$ python ss.py
i= 2 m= 1 min j, m 1 1
i= 2 m= 2 min j, m 2 2
i= 2 m= 3 min j, m 2 3
i= 2 m= 4 min j, m 3 4
i= 2 m= 5 min j, m 5 5
i= 2 m= 6 min j, m 5 6
i= 2 m= 7 min j, m 5 7
i= 2 m= 8 min j, m 5 8
i= 3 m= 1 min j, m 1 1
i= 3 m= 2 min j, m 2 2
i= 3 m= 3 min j, m 3 3
i= 3 m= 4 min j, m 3 4
i= 3 m= 5 min j, m 5 5
i= 3 m= 6 min j, m 5 6
i= 3 m= 7 min j, m 5 7
i= 3 m= 8 min j, m 5 8
points:
[1, 2, 3, 4, 7, 8, 9, 10]
final matrix:
    0    0    0    0    0    0    0    0    0
    0  0.0  0.5  2.0  5.0 21.2 38.83 58.86 82.0
    0  0.0  0.0  0.5  1.0  5.0  5.5  7.0 10.0
    0  0.0  0.0  0.0  0.5  1.0  1.5  3.0  6.0
MBP-di-Ciprian:HW3 mychro94$ _
```

The input is an array of 8 numbers. The final clusters will be: (1,2), (3,4) and (7,8,9,10). We need backtracing to find them, so we start from the position [3,8] of the matrix, that has value 6. We can see that the min values of j and m that generated 6 were 5 and 8 respectively. Notice that the indexes 5, ...8 denote the values of $6, 7, 8, 9$ that are the last cluster. We know that the previous value of i is 2, and that $j + 1 = 5$, so $j = 4$. Therefore we look at the minimum values

of the call D[2][4], that are 3 and 4, so the second cluster is formed by the input numbers at indexes 3 and 4, that are 3 and 4. For the same reasoning before, we know that D[1][2] is a base case, therefore the numbers 1 and 2 will form the third cluster.
Following there is another example.



```
[MBP-di-Ciprian:HW3 mychro94$ python ss.py
i= 2 m= 1 min j, m 1 1
i= 2 m= 2 min j, m 2 2
i= 2 m= 3 min j, m 2 3
i= 2 m= 4 min j, m 4 4
i= 2 m= 5 min j, m 4 5
i= 2 m= 6 min j, m 6 6
i= 2 m= 7 min j, m 6 7
i= 2 m= 8 min j, m 6 8
i= 3 m= 1 min j, m 1 1
i= 3 m= 2 min j, m 2 2
i= 3 m= 3 min j, m 3 3
i= 3 m= 4 min j, m 4 4
i= 3 m= 5 min j, m 5 5
i= 3 m= 6 min j, m 6 6
i= 3 m= 7 min j, m 6 7
i= 3 m= 8 min j, m 6 8
points:
[1, 2, 3, 10, 13, 27, 29, 30]
final matrix:
   0    0    0    0    0    0    0    0    0
   0  0.0  0.5  2.0 50.0 114.8 489.33 820.86 1099.88
   0  0.0  0.0  0.5  2.0  6.5 114.8 116.8 119.47
   0  0.0  0.0  0.0  0.5  2.0  6.5  8.5 11.17
MBP-di-Ciprian:HW3 mychro94$ 
```

Here the resulting clusters are: (1, 2, 3), (10, 13) and (27, 29, 30). The Python file can be run with *python hw3ex2-1.py* command from the terminal.

Exercise 2.2

For solving this problem we can first try to prove the hint, that is adding two centers, for an appropriate value of the centers, the cost decreases by a factor of 3/4. For simplicity, assume having just two points, $+x$ and $-x$. The cost P in this situation will be:
$||P|| = (-x-0)^2 + (x-0)^2 = 2x^2$
We now choose a center l, defined as $l = x/2$, so basically we are going to put the center l in between 0 and x, so for sure x will be now closer to l than to 0. Let's say l is added between 0 and the positive number x. The new cost will be:
$||P'|| = (-x-0)^2 + (x-l)^2 = \frac{5}{4}x^2$
We can now observe that the difference between the two costs is of $3/4x^2$. So in this case, of just two points and only a center being added, we can observe that the cost decreases by a 3/4 factor. This was a case in which we had only two points, but generally $l = x/2$ is not the optimal choice.

The optimal choice of l is:
$l = \frac{1}{|D|} \sum_{x \in D} x^2$
that is the mean of the points in D, where D is a set of points. In the case in which the cardinality of D is 1, then the center will be added exactly on the point, that is the optimal choice of the center in that case. What about 2 points

in D? We can prove that the best choice of l remains the mean of the two points. In fact, we will show that values that are bigger or smaller than the mean are returning a bigger cost.
Proof We define μ as the mean of the two points:
$\mu = \frac{x_1 + x_2}{2}$
where x_1 and x_2 are the points to be considered. Therefore, the cost in this case will be:
$||P||^2 = (x_1 - \mu)^2 + (x_2 - \mu)^2$
We proceed by contradiction, and we say that there is a better center $\mu' = \mu \pm \varepsilon$, where ε is a very small quantity. We can define P' as being the cost considering μ':
$||P'||^2 = (x_1 - \mu')^2 + (x_2 - \mu')^2$
and by contradiction we affirm that $||P'||^2 < ||P||^2$:
$(x_1 - \mu')^2 + (x_2 - \mu')^2 < (x_1 - \mu)^2 + (x_2 - \mu)^2$
after the computations we will have:
$2(\mu')^2 - 2\mu^2 < \pm 2\varepsilon(x_1 + x_2)$
and now for simplicity we can consider just the case in which μ' is greater than μ, because the other case is equivalent. We can simplify:
$\varepsilon + 2\mu < x_1 + x_1$, we replace μ and get:
$x_1 + x_2 + \varepsilon < x1 + x_2$, and since ε is positive this is a contradiction, and means that it is not true that $||P'||^2 < ||P||^2$. This being said, we can affirm that the mean of the numbers is the optimal choice for a new center to add to a set of numbers. This can be proven also for the case in which the number of points is greater than 2.

Let's get now to the general case of adding $O(1/\varepsilon)$ nodes. Before to proceed with the demonstration we have to make clear other assumptions. The proof we want to make is by induction, therefore we need to have at some step t the optimal cost until that step. Therefore, since t centers have been added, we need to figure out how these centers have been added. By defining the center as above, we will add always the same centers, and the cost won't decrease. Therefore I will define an algorithm that, each time a center has to be added, computes the value of the center.
We will proceed by intervals. We define the set I as being the union between the set C of centers and the maximum point and the minimum point belonging to the set of points P. Assume I being ordered in a non descending way. Then the center l will be found with the following algorithm:

```
interval = (0,0)
for i in I:
    size = $I[i+1] - I[i]$
    if size > max:
        max = size
        interval = (i, i+1)
l = mean(numbers in X between i and i+1 centers)
add l to set I
```

With this reasoning we will work on intervals each time we have to add a center, getting the largest current interval, computing the mean of the points belonging to the interval and adding the center inside the interval. We know in this way

that the mean is the optimal choice for a new center.
We know that adding a center, the cost does not increase, because with an optimal choice of the center given by the algorithm above, in the worst case no point will be closer to the new center(very particular case), but very often will happen that at least one point will be closer to the new center.
First of all we need to understand what changes from one step to another, in the sense that we want to know how much the cost decreases from the step t to the step t+1. As said before, at step t we have t centers already present, plus the center in 0. Adding a new center μ we can assume in the worst case that a point x_f will be closer to this new center than to all the others. So, what will be the cost of the point x_f? We call μ' the closest center after μ, so the cost paid by the point x_f was:
$c_{xf}^{\mu'} = (x - \mu')^2$,
and, after adding μ, that is closer to x_f, the cost will be:
$c_{xf}^{\mu} = (x - \mu)^2$, and therefore $c_{xf}^{\mu} < c_{xf}^{\mu'}$ by a quantity:
$2x(\mu - \mu') + (\mu' - \mu)(\mu' + \mu)$
Recall that $b^2 - a^2 = (b - a)(b + a)$, and this equation was used to find the quantity above.

Now for simplicity we can assume that each center t in the set of centers C, including the center in 0, has its own set of points, and these points are the closest points to the center t. Therefore we can consider $S_0, ..., S_t$ sets of points. We define the original cost as being the cost when having in C only the center 0 as:
$||P^0||^2 = \sum_{x \in P} x^2$, where X is the set of all points.
Since we are at step t (also t is the number of added centers), we know that:
$||P^t||^2 \leqslant \dfrac{||P^0||^2}{t}$
We can write P^t also in the following way:
$||P^t||^2 = \sum_{\mu' \in C} \sum_{x \in S_{\mu'}} (x - \mu')^2$
that is the sum of all the costs of the points wrt their closest center. Adding another center, let's call it μ, in the worst case just one point belonging to some other center will be now closer to μ. We need now to prove that:
$||P^{t+1}||^2 = ||P^t||^2 - 2x(\mu - \mu') + (\mu' - \mu)(\mu' + \mu) \leqslant \dfrac{||P^0||^2}{t+1}$
where x is the point that before was closer to μ' but now it is closer to the new center added, μ. What we are saying is that the new cost is equal to the previous one, decreased by the *gain* of adding a new center closer to a point that first was belonging to the set of points of the center μ'.
We can observe that adding a center that is the closest one to some point in P, then the cost for sure decreases.