PROJECT REPORT ASSIGNMENT 3

Introduction to Machine Learning and Data Mining

Spring 2018

Lino Valdovinos

861300001

CS-171

INTRODUCTION K-means Clustering

For this assignment I got to work with the Iris dataset again, but this time I had to ignore classification and implemented k-means to determine the number of centroids, and K-means++ to select better initial centroids.

PROCEDURE

K-means

1. Random initialized centroids k from the data.
2. Use distance function L2(Euclidean) to cluster every point to the nearest centroid. Additionally, give the data points in a cluster an arbitrary assignment.
3. Calculate ncw centroids based on the mean values per feature of the data in a given cluster.
4. Repeat (2) and (3) util the assignment vector stop changing.

K-means++

1. Pick one random initialized centroid.
2. Compute the distance from (1) to every datapoint and square it and store it in a vector. (Dist)
3. Sort the distance vector.
4. Calculate the sum of all those distances. (ALL_SUM)
5. Then a new P vector the following way
    a. P[i]= SIGMA(Dist[i]) / (ALL_SUM) from 0 to i
6. The P vector ranges from [0-1]
7. Get a random number R from [0-1]
8. Then get new centroid the following way
    a. P[j-1] < R <= P[j] then then the new centroid is data[j]
9. For more centroid repeat (1) -(8) but the use the distance to the nearest centroid.

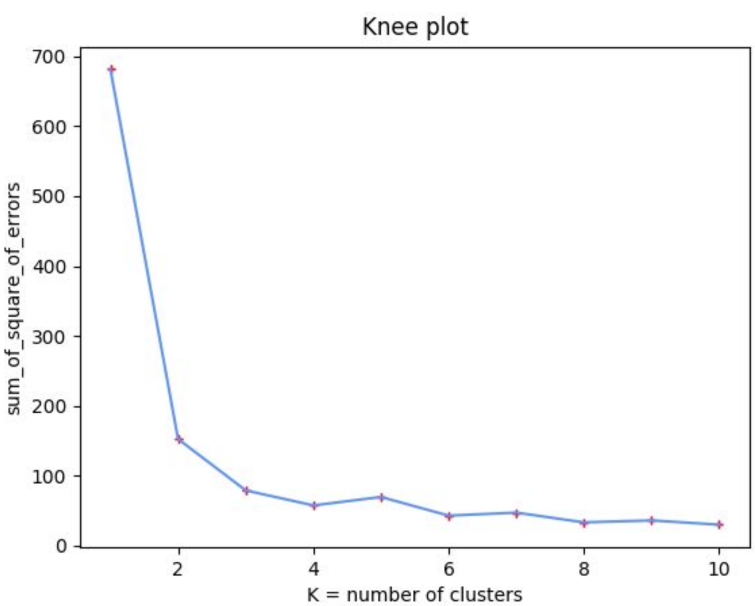
DATA

For k = 3 error from the sum of square of errors is 78.851441426146.

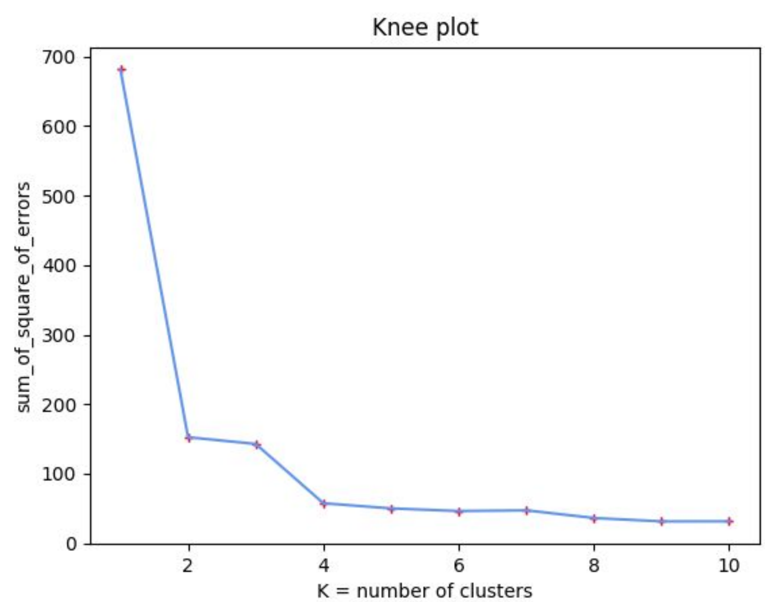
This was calculated using L2 distance squared.

I do not allow repeated random values. Therefore the index of any number of initial centroid will never be the same. However, I do not take into account if two or more different data points have the exact same values.

Knee-plots



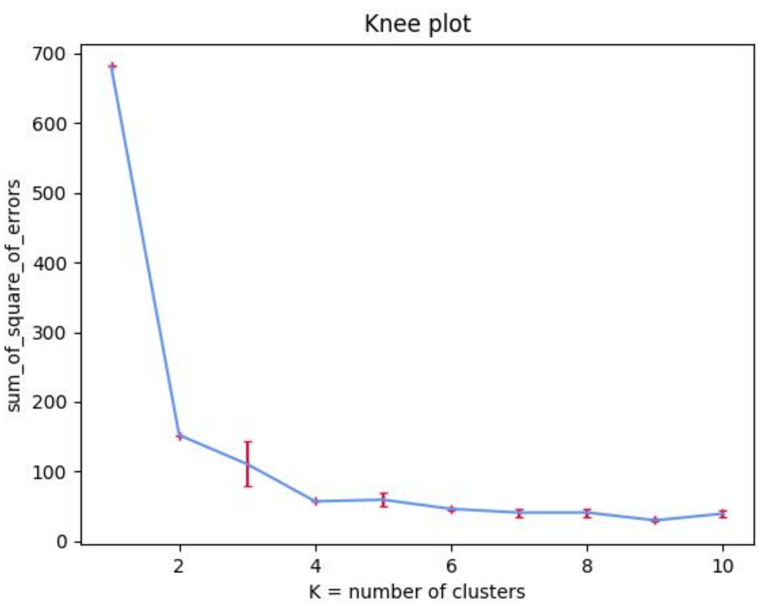
Random

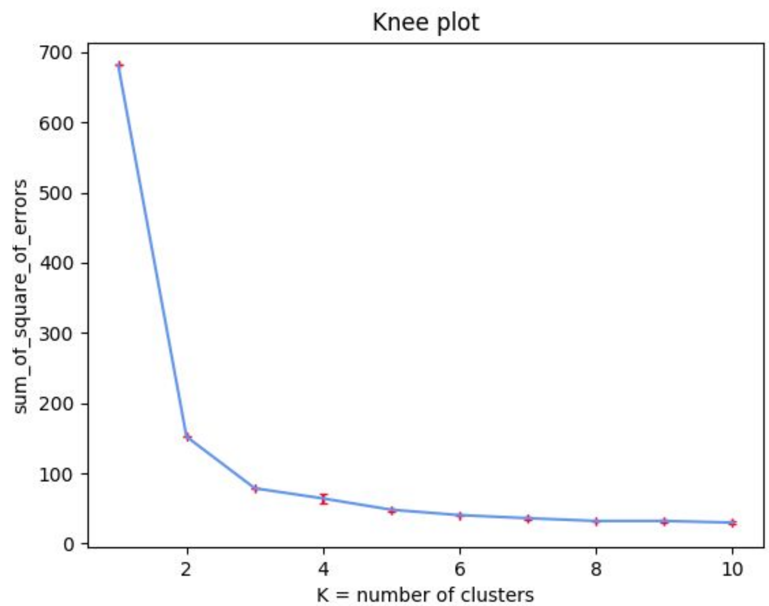


Random

Basic knee-plots. Only one iteration for both. Form the left image is seems that 3 is the correct number of cluster,which agrees with the number of classes. In addition, this is and example in which random initialization of the centroids actually works properly. However the example to the right is an example of random initialization of centroids not working properly. The error is just slightly better from k = 2 to k = 3 most likely do to that fact the two initial centroids were from the same class. I would not consider this graph for approximating the correct number of centroid since the results are unstable.

Max_iter = 2



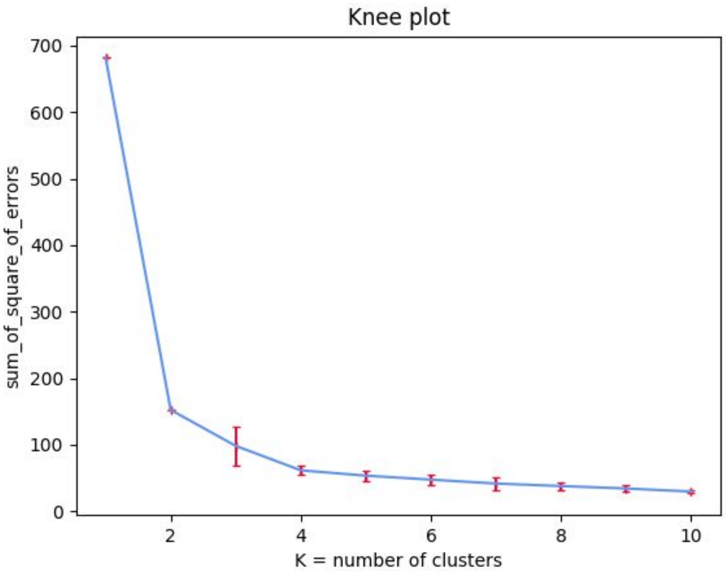


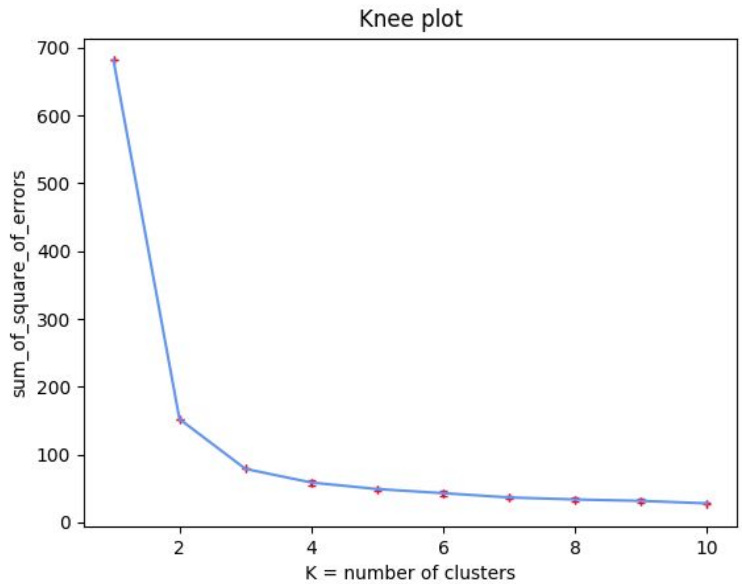
Random

k++

With two iteration we already get a better picture of knee plot. In both cases it seem pretty clear that 3 is that correct number of centroids matching with our classes. Notice that the standard of deviation on the left graph for k = 3 is high again do to the fact that random initialization could lead to bad results.

Max_iter = 10

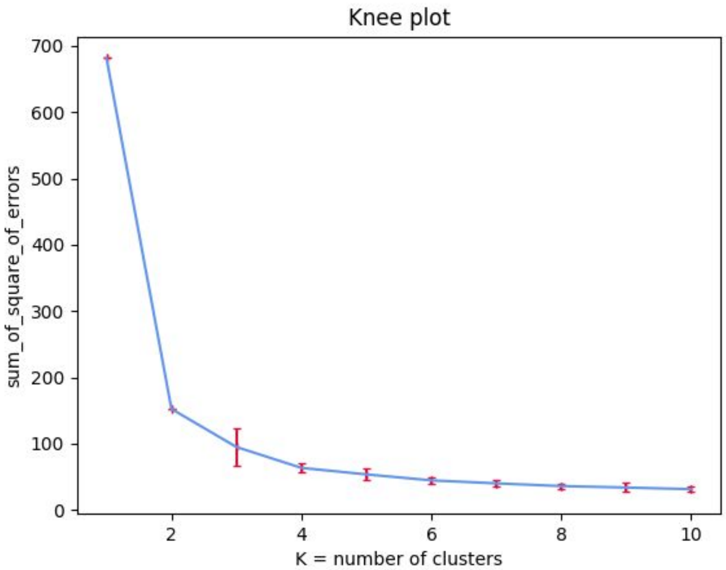




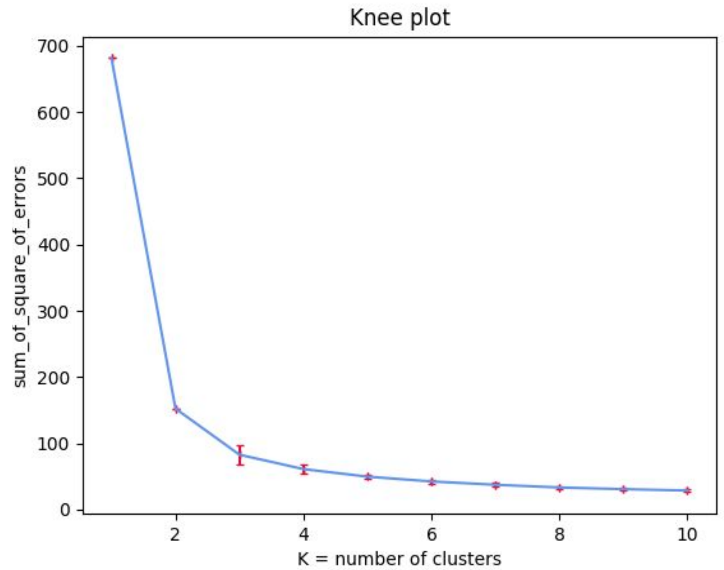
Random K++

After 10 interactions the knee plots smooths out. The main takeaway from these two graphs is that k-means++ does a much better job at selecting the initial centroids, since the standard of deviation for all 1-10 is zero.

Max_iter = 100



Random



K++

Similar to the 10 iterations, after 100 iterations the knee plots are really smooth. Although this time we can see that k-means++ is not perfect since, the standard of deviation for k = 3 and k= 4 are not zero. However it still proves to be far more effective then just random selection.

Results for Top 3- data points per cluster classified

3 Multiple runs to check for randomness

Initial centroid	Initial centroid	Initial centroid
[[5.7 4.4 1.5 0.4]	[[6.3 3.3 6. 2.5]	[[6.4 2.8 5.6 2.1]
[6.9 3.1 5.1 2.3]	[6.3 2.8 5.1 1.5]	[4.8 3. 1.4 0.1]
[4.5 2.3 1.3 0.3]]	[5.7 2.9 4.2 1.3]]	[4.7 3.2 1.6 0.2]]
Cluster: 1	Cluster: 1	Cluster: 1
['Iris-setosa']	['Iris-virginica']	['Iris-virginica']
['Iris-setosa']	['Iris-virginica']	['Iris-virginica']
['Iris-setosa']	['Iris-virginica']	['Iris-virginica']
Cluster: 2	Cluster: 2	Cluster: 2
['Iris-virginica']	['Iris-versicolor']	['Iris-setosa']
['Iris-virginica']	['Iris-versicolor']	['Iris-setosa']
['Iris-virginica']	['Iris-versicolor']	['Iris-setosa']
Cluster: 3	Cluster: 3	Cluster: 3
['Iris-setosa']	['Iris-setosa']	['Iris-setosa']
['Iris-setosa']	['Iris-setosa']	['Iris-setosa']
['Iris-setosa']	['Iris-setosa']	['Iris-setosa']

Notice that the first run two of the centroids fall in the same class and as a result cluster 1 and cluster 2 top data point are in the same class. Run number two outputs perfect values since the initial centroids fall under different classes. However, in the last run, two of the centroids are in the same class and the result the cluster label the same thing.

CONCLUSION

K-means proofs to be an effective algorithm at determining the total number of clusters in the data. In addition, I felt like it is intuitive as well as fun to implement.

K-means++ also proves to be far better than just random initialization .

REFERENCES

Class slides

http://www.cs.ucr.edu/~epapalex/teaching/171_S18/index.html#schedule

Matplotlib documentation

https://matplotlib.org/index.html

Numpy documentation

https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.array.html

Python documentation

https://docs.python.org/2/library

Iris data-set

https://archive.ics.uci.edu/ml/datasets/Iris

Late Days

0 used for this assignment

0 used so far