Clustering Analysis

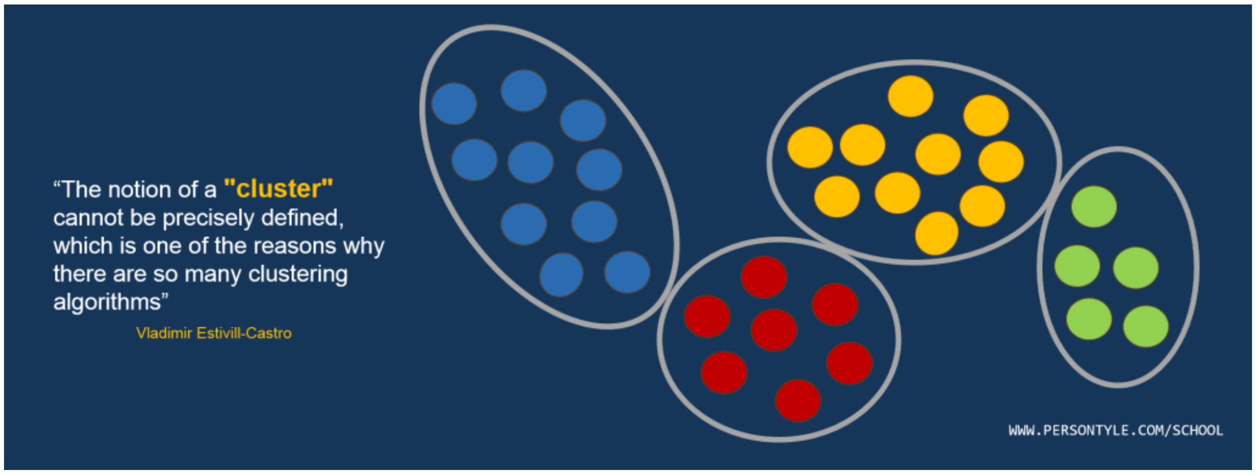

"The notion of a "cluster" cannot be precisely defined, which is one of the reasons why there are so many clustering algorithms"
Vladimir Estivill-Castro
WWW.PERSONTYLE.COM/SCHOOL

Table of Contents

Clustering Analysis

While *prediction* and *inference* are the primary reasons to use statistical models, there are some situations where it is not possible to apply the same principles. Some data sets don't have already-defined labels or answers to train models and compare results. We can, nonetheless, explore this data sets, and find patterns and/or commonalities among its items. We are now entering the realm of *Unsupervised learning*: the discovery of structures in unlabeled data.

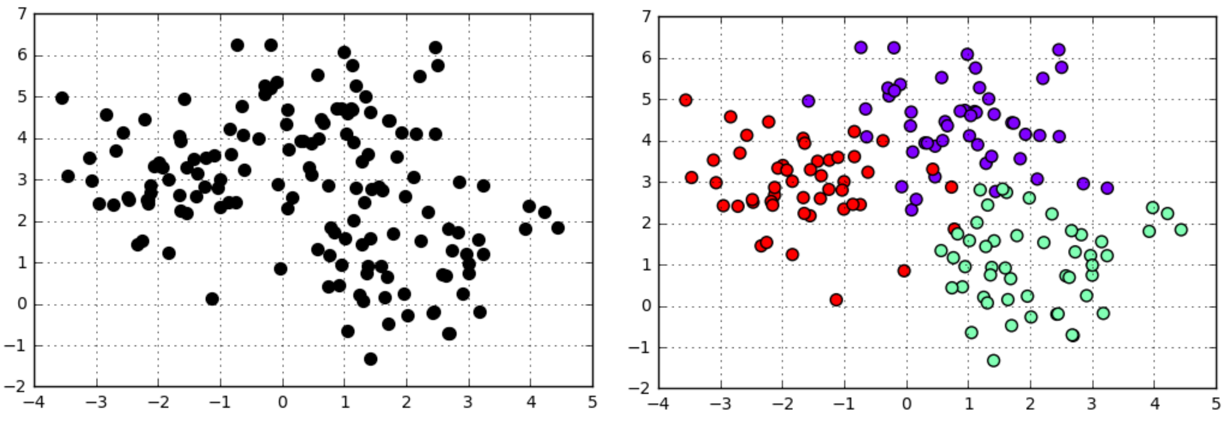
There are many examples where exploratory data technics are used in real life. For example, retail businesses group their customers by common attributes and target them with different marketing campaigns. Medical patients are usually classified by common health and biological markers to receive different treatments and look for disease causality. It is important to note that these classification results (better known as *clusters*) don't contain a "right or wrong" component since there is no "true categories" to compare against. However, there are models better suited for different types of data and some ways to optimize our results.

Throughout this document, I will explore the famous ***K-means*** algorithm, I will offer a method to better select and the number of clusters, and finally I will test the quality of the selection.

Section 1: The K-means Algorithm - Definition

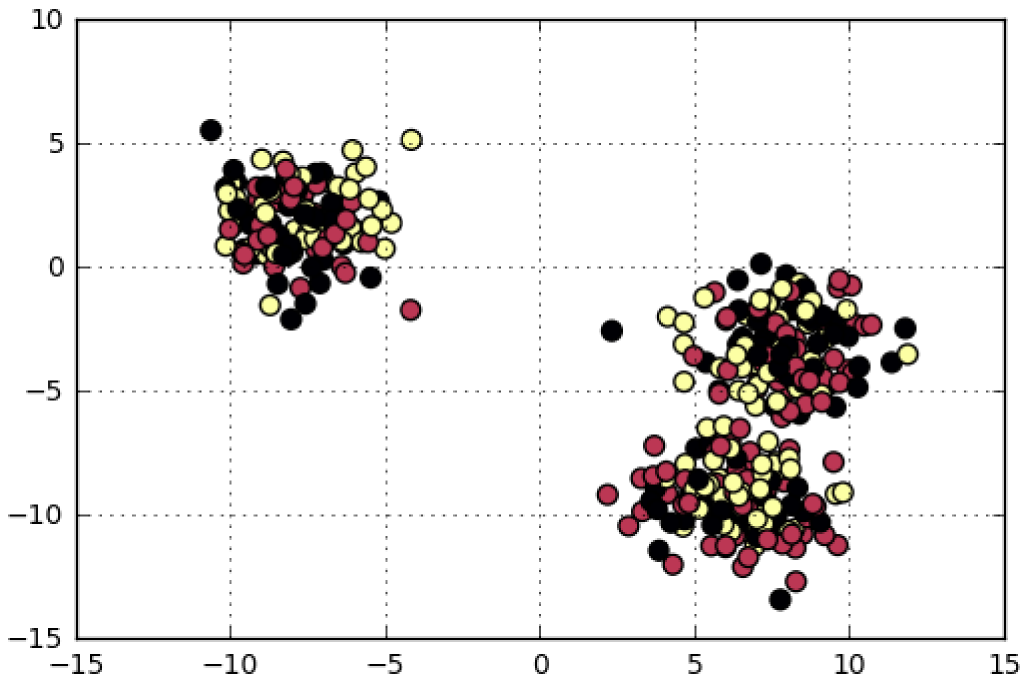
This simple algorithm is commonly used to create clusters by *proximity*. It is easy to implement and computationally faster than others algorithms in the same class. It belongs to a group called ***Prototype-based clustering*** since there is a prototype that represents the cluster, either the average (centroid) or the most frequent (medoid) item.

This is a visual representation of 150 data points grouped together by K-means. Each color represents a cluster. There is no specific order (hierarchy) or labels applied to clusters:

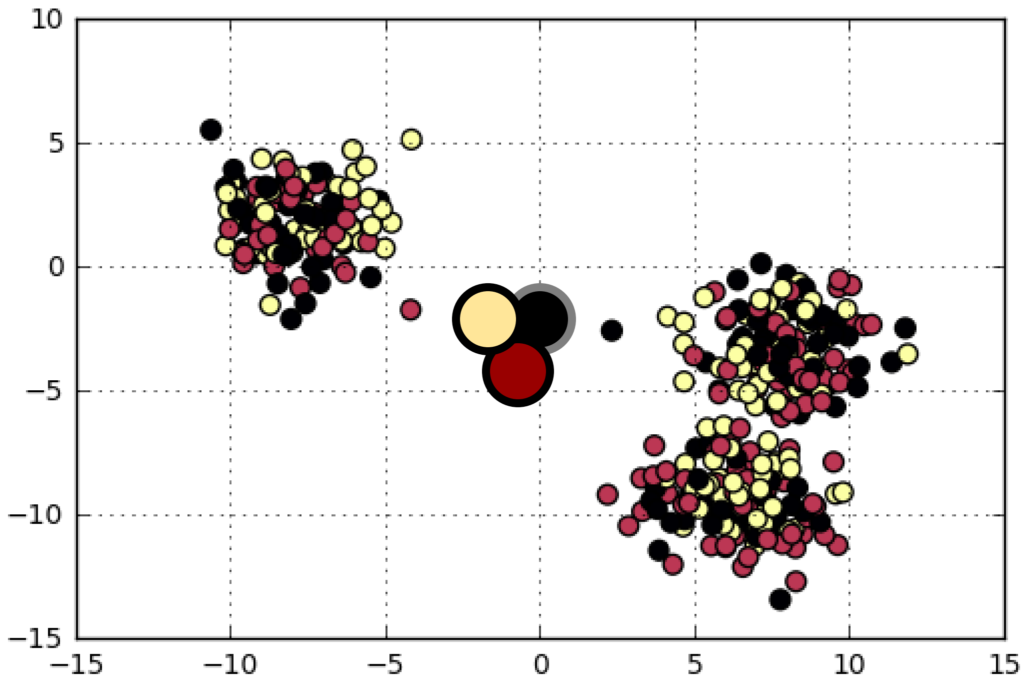


How are data points classified? It is important to note that this function doesn't come up with the best number of classes (*k*); we must specify the number of classes we want beforehand. With that value, the function compute the following 4 steps:

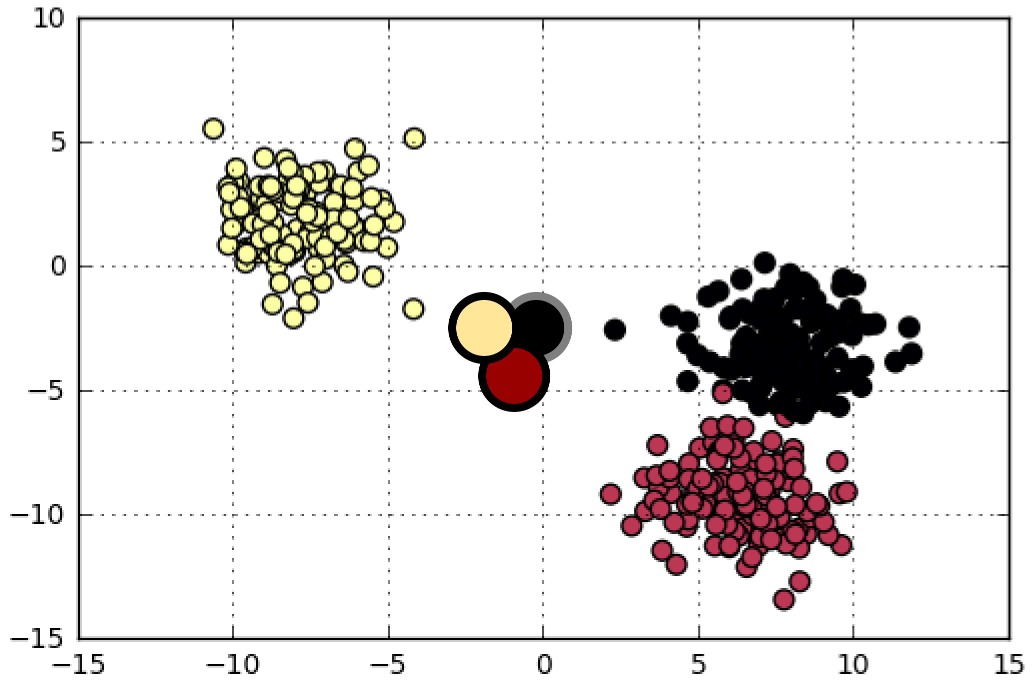
a) Every single data point is randomly assigned to a cluster.



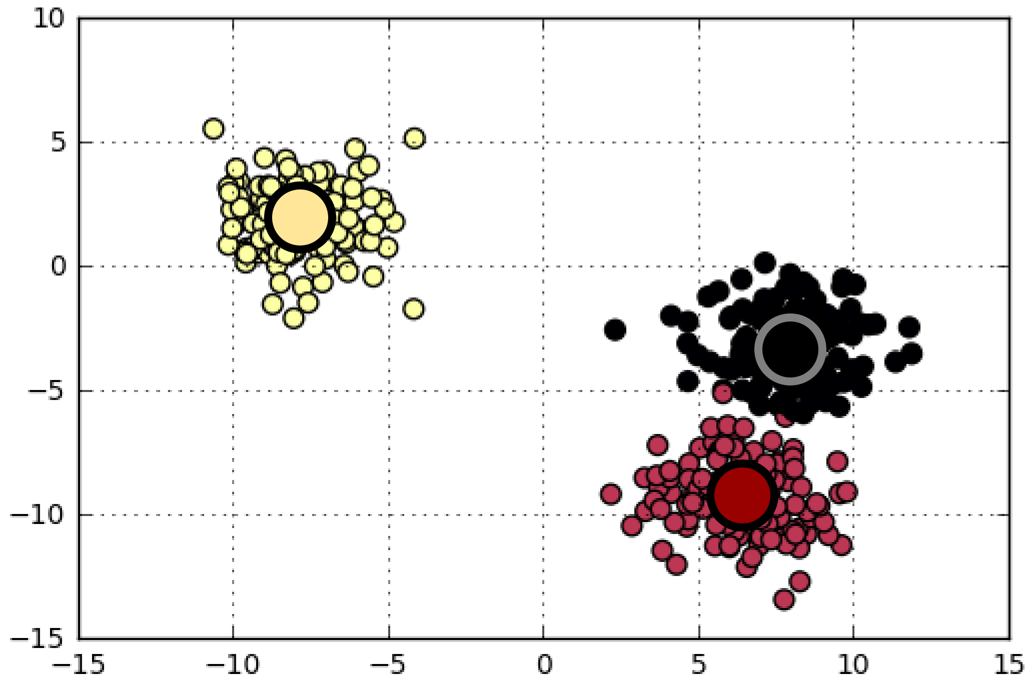
b) The cluster *centroids* -the mean of the data points in the clusters- are found.



c) Data points are assigned to the same cluster as the closest centroid (by squared *Euclidean distance*).



d) Steps b and c will iterate until the function finds no more changes in the clusters.



K-means can also be described as an *optimization function* where the goal is to minimize the total distance between the *centroids* and the data points around them (local distance). The objective formula is:

$$\underset{\mathbf{S}}{\arg\min} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \left\| \mathbf{x} - \boldsymbol{\mu}_i \right\|^2$$

Where,

$\left\| \mathbf{x} - \boldsymbol{\mu}_i \right\|^2$ is the squared Euclidean distance between the data points and their means.

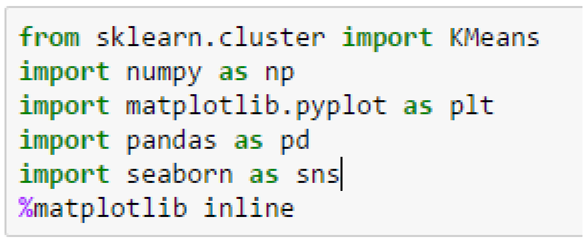
$\sum_{\mathbf{x} \in S_i}$ represents the number of observations within each set.

$\sum_{i=1}^{k}$ represents the number of partitions (clusters).

Section 2: The K-means Algorithm – Application in Python

I will apply the algorithm on a real data set (College.csv) using Python (with Jupyter Notebook). This data set has 17 numeric variables, 1 categorical variable, and 777 instances. Among the numerical variables there are measurements as: acceptance rate, number of students enrolled, tuition, book costs, graduation rate, number of faculty with PhD, etc. The categorical value indicates if the university is private or not. The idea is to observe how K-means classifies this data points in 2 clusters: private or public university.

Again, it happens that for this data set we have the “true” category -private or public- and we can readily decide that we want to run the function with k=2 (2 clusters), but in real life we rarely use this function having this pre-knowledge. Number of clusters will depend on specific goals, or expertise, or by discovering obvious patterns in the data. It should be pointed, however, that the number of clusters chosen is of utmost importance since our main objective is to represent real sets within the data. As with any other data mining technique, we could fall into the trap of over-classification due to error produced by randomness (*noise*).



```python
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
%matplotlib inline
```

```
college = pd.read_csv("college.csv",index_col=0)
```

```
college.head(4)
```

	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Unde
Abilene Christian University	Yes	1660	1232	721	23	52	2885	537
Adelphi University	Yes	2186	1924	512	16	29	2683	1227
Adrian College	Yes	1428	1097	336	22	50	1036	99
Agnes Scott College	Yes	417	349	137	60	89	510	63

```
college.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 777 entries, Abilene Christian University to Yo
a
Data columns (total 18 columns):
Private        777 non-null object
Apps           777 non-null int64
Accept         777 non-null int64
Enroll         777 non-null int64
Top10perc      777 non-null int64
Top25perc      777 non-null int64
F.Undergrad    777 non-null int64
P.Undergrad    777 non-null int64
Outstate       777 non-null int64
Room.Board     777 non-null int64
Books          777 non-null int64
Personal       777 non-null int64
PhD            777 non-null int64
Terminal       777 non-null int64
S.F.Ratio      777 non-null float64
perc.alumni    777 non-null int64
Expend         777 non-null int64
Grad.Rate      777 non-null int64
```

```
from sklearn.cluster import KMeans

km = KMeans (n_clusters = 2, init = 'k-means++', n_init = 20,
             max_iter = 300, tol = 0.0001, random_state = 0)
# ~ PARAMATERS ~
# n_clusters: for k=2
# init: 'random' or 'k-means++'(used to improve initial assignments)
# n_init:to run the algorith 20 times and choose the iteration
#            with the lowest local distance (from centroids)
# max_iter: to restrict the number if iterations before
#            convergence (to improve processing time )
# tol: level of tolerance (distance) to declare convergace
# random_state: to produce consitante results given that
#                KMeans starts with ramdonly assignments

model_km = km.fit_predict(college.iloc[:,1:])
```

```
# Final model with classifications as 0s and 1s
model_km
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
# Now, after havnig a list of assigments (model_km) I want
# to compare those results with the "true" values from college['Private'].
# I will create a new column ('cluster') to assign values 1 and 0 to
# "Yes" and "No".

def conv(private):
    if private =='Yes': return 1
    else: return 0

college['cluster'] = college['Private'].apply(conv)
```

```
from sklearn.metrics import confusion_matrix,classification_report
print(confusion_matrix(college['cluster'],model_km))
print('\n')
print(classification_report(college['cluster'],model_km,
                            target_names=['*Yes*','*No*']))
```

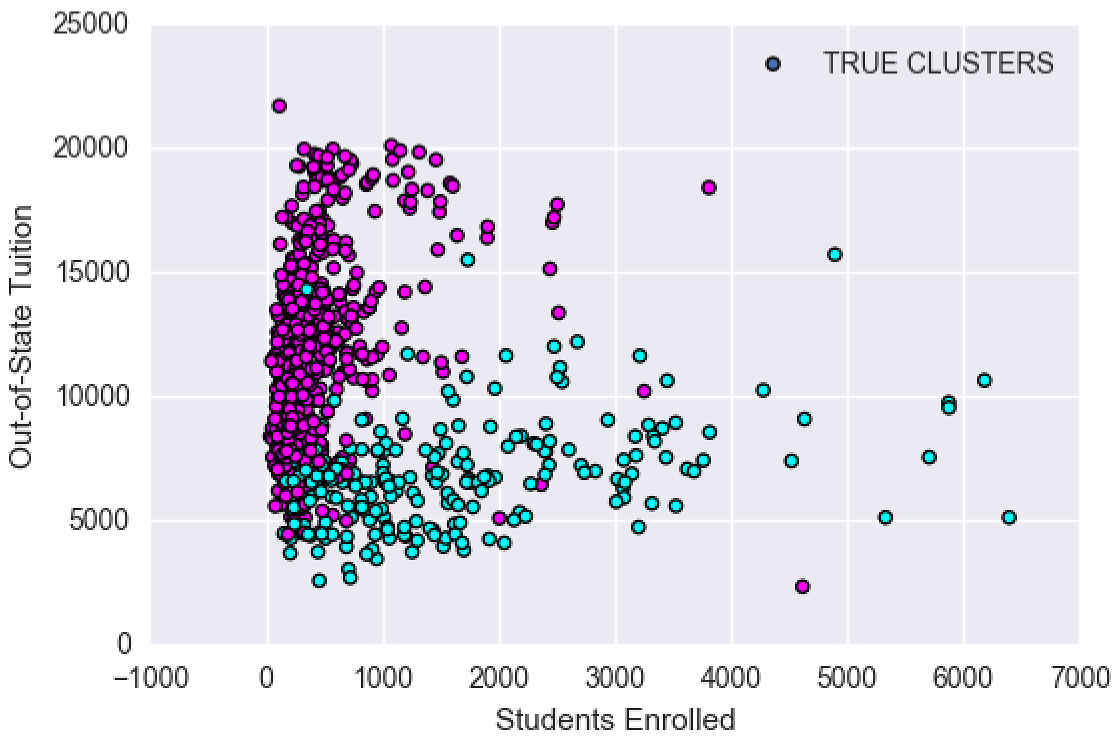
```
[[138  74]
 [531  34]]
```

```
              precision    recall  f1-score   support

      *Yes*       0.21      0.65      0.31       212
       *No*       0.31      0.06      0.10       565

avg / total       0.29      0.22      0.16       777
```

```
# If we compare there is a 65% match (predictions?) for 'Yes'
# and only 6% match for 'No'.
# These results can't be treated as predictions! They can't
# be used to determine how good or bad our model is since
# we are not concerned with 'prediction' but with 'clustering'.
```

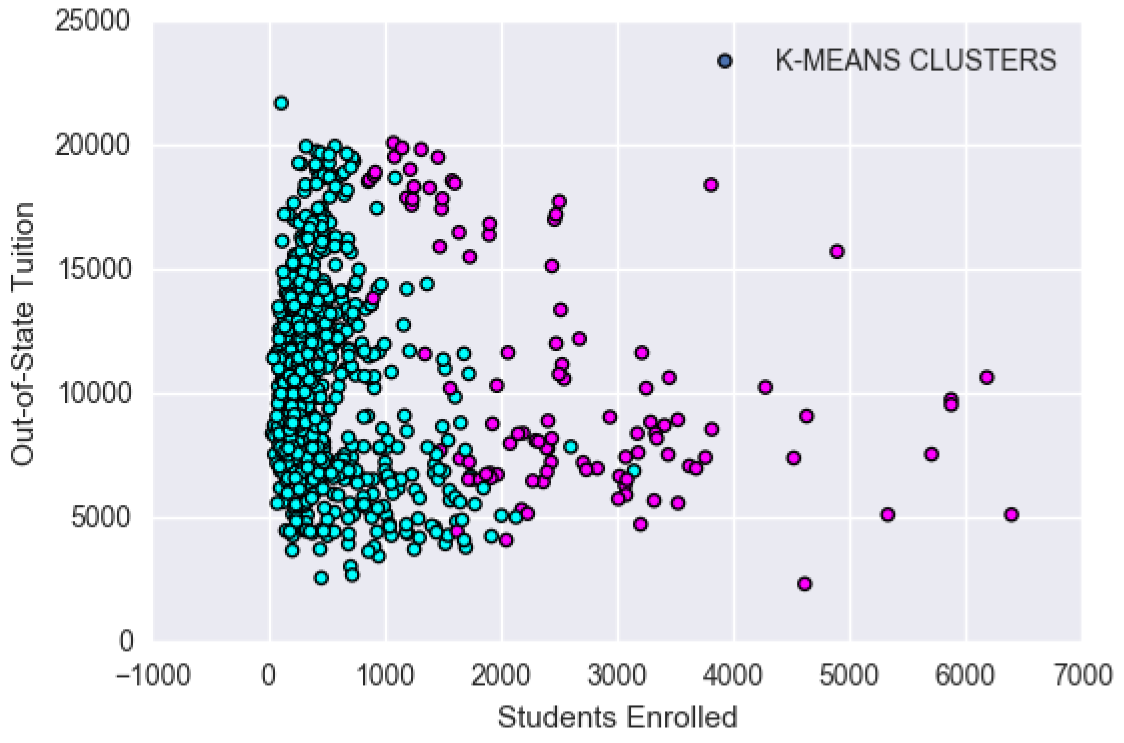
```
# Now, let's visually compared our 'true' clusters with K-Means 'clusters'.

plt.scatter(college['Enroll'],college['Outstate'],
            c=college['cluster'],cmap='cool',linewidths=1,
            label='TRUE CLUSTERS')
plt.xlabel('Students Enrolled')
plt.ylabel('Out-of-State Tuition')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x280d4730b70>
```



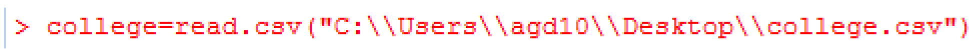
```
# It should be noted that we are only comparing clusters in 2 dimension.
# It is impossible to visually assess the quality of our clusters
# if we were to plot including all variables (all dimensions).

plt.scatter(college['Enroll'],college['Outstate'],
            c=model_km,cmap='cool',linewidths=1,
            label='K-MEANS CLUSTERS')
plt.xlabel('Students Enrolled')
plt.ylabel('Out-of-State Tuition')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x280d4479e10>
```

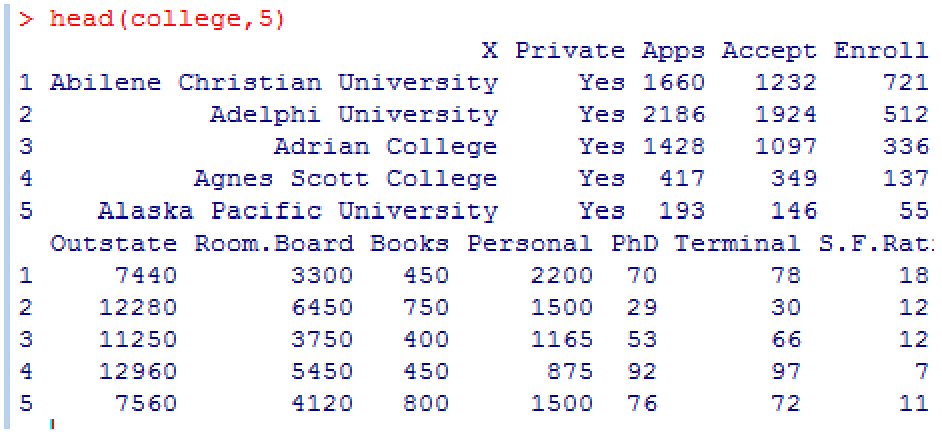


Section 3: The K-means Algorithm – Application in R

Running the same algorithm in R is a straight forward process; there is no need to install any package, function is already built-in. I will use the same “college.csv” file:

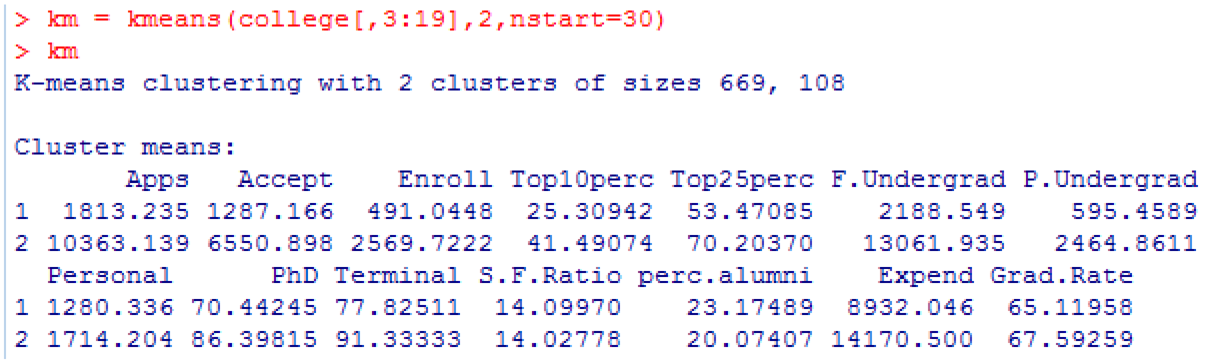


```
> college=read.csv("C:\\Users\\agd10\\Desktop\\college.csv")
```

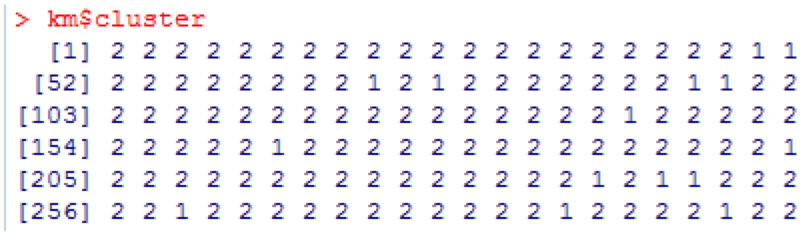


```
> head(college,5)
                             X Private Apps Accept Enroll
1 Abilene Christian University     Yes 1660   1232    721
2           Adelphi University     Yes 2186   1924    512
3               Adrian College     Yes 1428   1097    336
4          Agnes Scott College     Yes  417    349    137
5    Alaska Pacific University     Yes  193    146     55
  Outstate Room.Board Books Personal PhD Terminal S.F.Rat
1     7440       3300   450     2200  70       78      18
2    12280       6450   750     1500  29       30      12
3    11250       3750   400     1165  53       66      12
4    12960       5450   450      875  92       97       7
5     7560       4120   800     1500  76       72      11
```

I am calling the function with columns [3] to [19] (numerical values only), with k=2, and with 30 iterations (to choose iteration with less *local distance*):

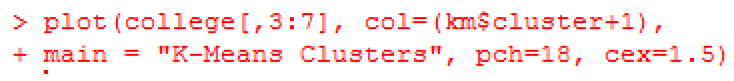


```
> km = kmeans(college[,3:19],2,nstart=30)
> km
K-means clustering with 2 clusters of sizes 669, 108

Cluster means:
      Apps   Accept    Enroll Top10perc Top25perc F.Undergrad P.Undergrad
1  1813.235 1287.166 491.0448  25.30942  53.47085    2188.549    595.4589
2 10363.139 6550.898 2569.7222  41.49074  70.20370   13061.935   2464.8611
  Personal      PhD Terminal S.F.Ratio perc.alumni    Expend Grad.Rate
1 1280.336 70.44245 77.82511  14.09970    23.17489  8932.046  65.11958
2 1714.204 86.39815 91.33333  14.02778    20.07407 14170.500  67.59259
```

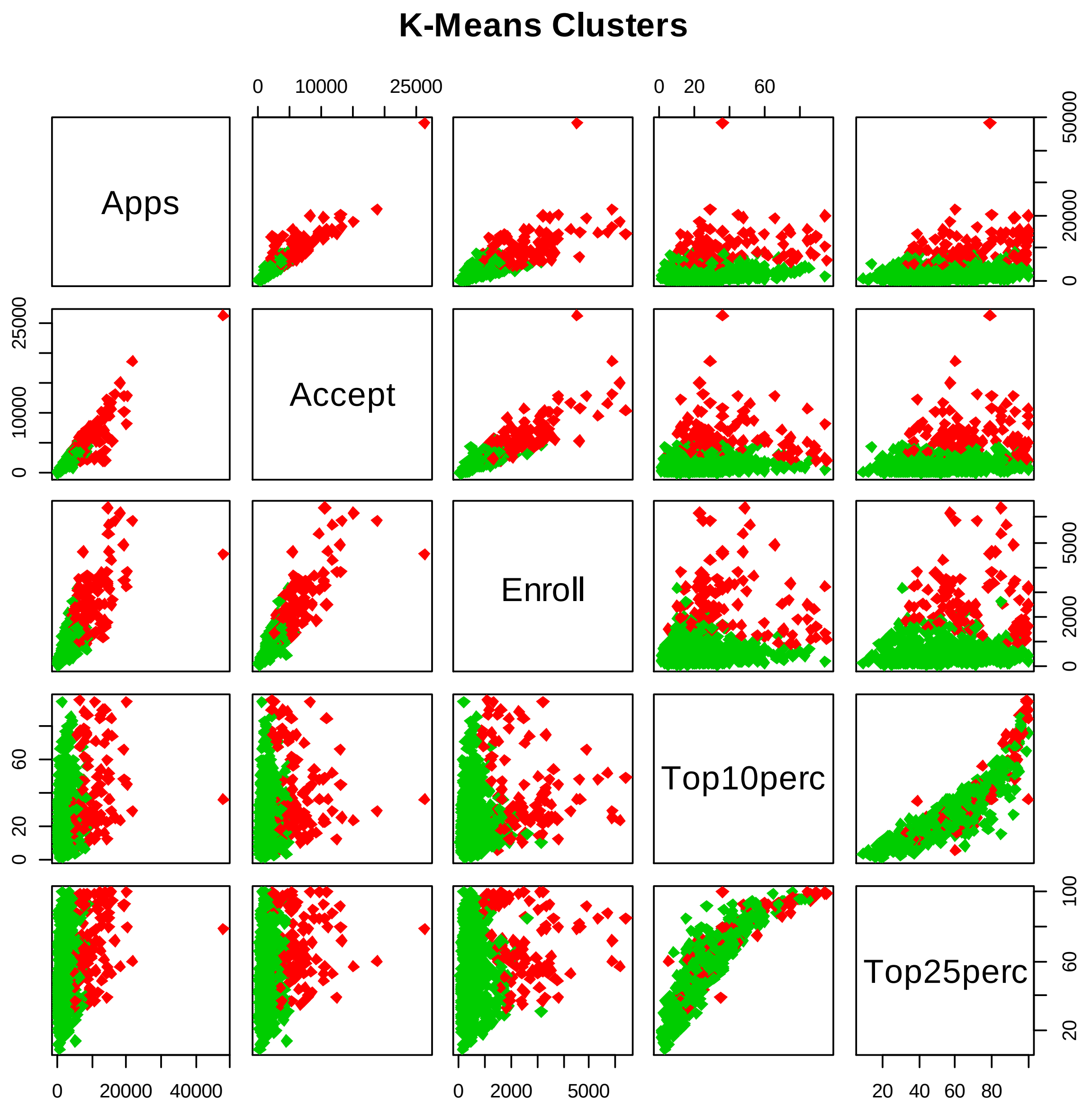


```
> km$cluster
  [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1
 [52] 2 2 2 2 2 2 2 2 1 2 1 2 2 2 2 2 2 2 1 1 2 2
[103] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2
[154] 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1
[205] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 1 2 2 2
[256] 2 2 1 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 1 2 2
```

Finally, I will plot the results to see how the algorithm groups together different values:



```
> plot(college[,3:7], col=(km$cluster+1),
+ main = "K-Means Clusters", pch=18, cex=1.5)
```

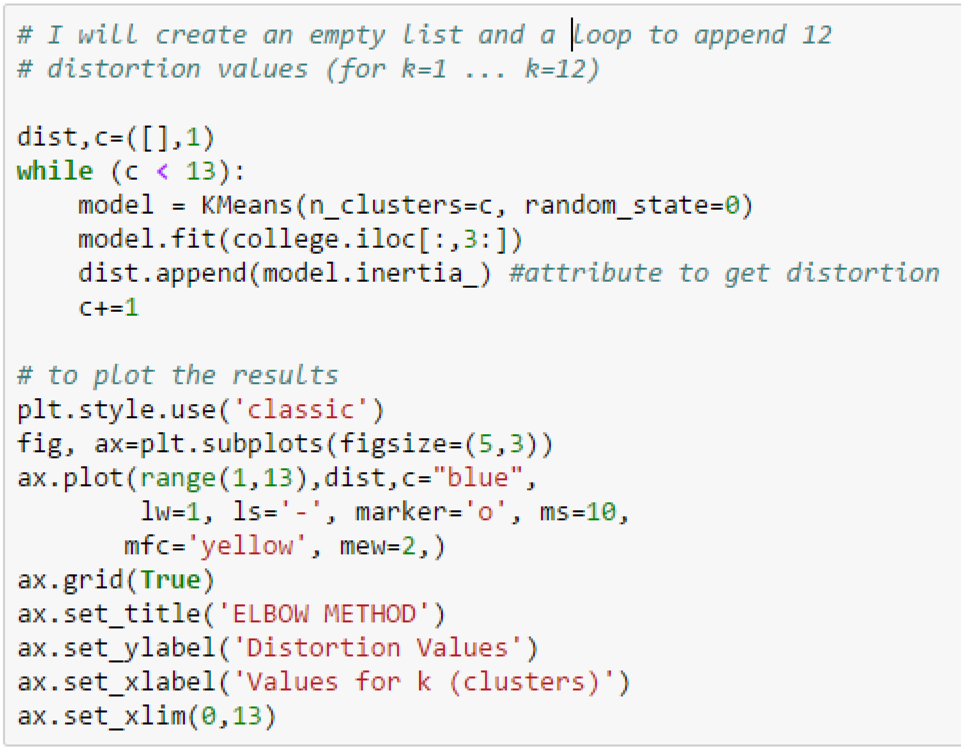


Section 4: How to select the number of clusters

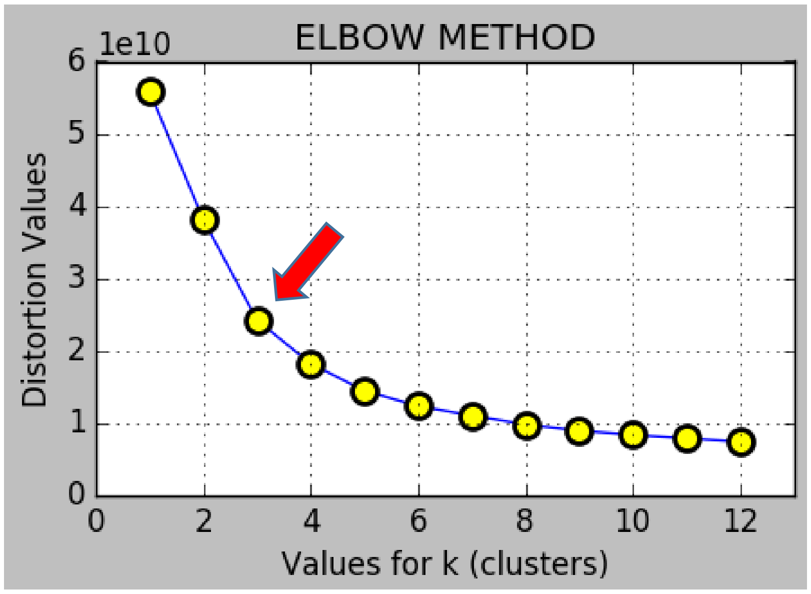
How can this model be of any use if we don't have the correct number of clusters beforehand? If we can arbitrarily choose 3 or 20 clusters, then why do we use this algorithm at all?

The truth is that we never "arbitrarily" select the numbers of clusters; at least not if we expect useful results. Above all, sound judgment must be applied by professionals with expertise in each specific field or industry where the data is explored. It is also possible to group by *natural* clusters that arise when the data is analyzed and/or visualized. Also, fortunately, there are some numerical tests that we can take to *optimize* the number of clusters. They work by taking intrinsic measurements from within the data and comparing the results.

Remember, what is the main goal of the K-Means function? It is to reduce the *local distance* or the sum of the squared Euclidean distances (or errors) from each sample point to their centroids. This measurement Is better known as *distortion*. And, there exists a positive relationship between the *distortion* and the *relevance* (cluster size) of your clusters. Your goal is to reduce distortion but not so much at the expense of relevance. There is however, a middle ground, a balanced area for k, where distortion is greatly reduced, and relevance is not largely compromised. To find this point we use the famous *elbow method*. The idea is simple: we choose the point where the elbow-shaped line is not steadily decreasing anymore.



```
# I will create an empty list and a loop to append 12
# distortion values (for k=1 ... k=12)

dist,c=([],1)
while (c < 13):
    model = KMeans(n_clusters=c, random_state=0)
    model.fit(college.iloc[:,3:])
    dist.append(model.inertia_) #attribute to get distortion
    c+=1

# to plot the results
plt.style.use('classic')
fig, ax=plt.subplots(figsize=(5,3))
ax.plot(range(1,13),dist,c="blue",
        lw=1, ls='-', marker='o', ms=10,
       mfc='yellow', mew=2,)
ax.grid(True)
ax.set_title('ELBOW METHOD')
ax.set_ylabel('Distortion Values')
ax.set_xlabel('Values for k (clusters)')
ax.set_xlim(0,13)
```



The Elbow plot shows that the optimum value for k is around 3. Passed that point the decreases in ***distortion*** are too small to keep losing more ***relevance*** in our clusters.

Section 5: Testing the quality of the clusters.

Having selected our number of clusters we can still measure their quality by using intrinsic measurements. One technique called *Silhouette Analysis* is used to measure *group cohesion,* or how closed together data points are in relation to their distance to other clusters. The following formula is used:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

Where,

a(i) = cluster cohesion measured as the average local distance from any observation **x**.

b(i) = cluster separation measured as the average distance from **x** and all other observation from the nearest cluster.

max {a(i) , b(i)} = the greater of the 2 previous values.

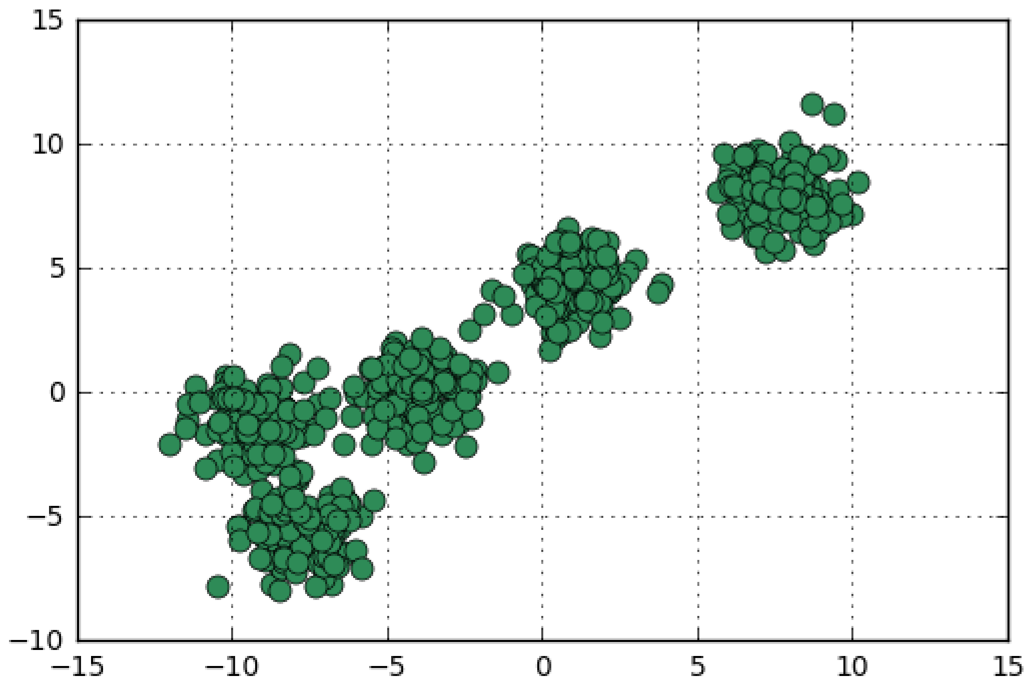
S(i) = the coefficient that goes from -1 to 1.

The interpretation of the coefficient is simple. The closer to 1 the more cohesion within the cluster. The closer to 0 the less cohesion, meaning the more dissimilar the observations within the group; hence, a weak cluster. If we get observations with values below 0 then there is an error in our algorithm since those observations have been totally misclassified. The ideal classification would show all clusters well above 0 and above the mean coefficient. Silhouette plots are used to better understand these relationships.

I will demonstrate how silhouette coefficients and plots are used. I will generate a random number of observations with well-defined means to make the example more dramatic:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import cm
%matplotlib inline
```

```
# Let's generate 600 random observations with normal distribution
# and 5 centers (means)
from sklearn.datasets import make_blobs

x, y = make_blobs(n_samples=600, n_features=2, centers=5,
                  cluster_std=1, random_state=3 )

fig, ax = plt.subplots()
ax.plot(x[:,0],x[:,1], marker='o',c='seagreen', ms=8, lw=0)
plt.grid()
```



```
# Let's creat a model to fit the data points
from sklearn.cluster import KMeans

# since we have generated our own data we know k=5
KM_model = KMeans(n_clusters=5, random_state=0)
KM_model.fit(x)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=5, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=0, tol=0.0001, verbose=0)
```

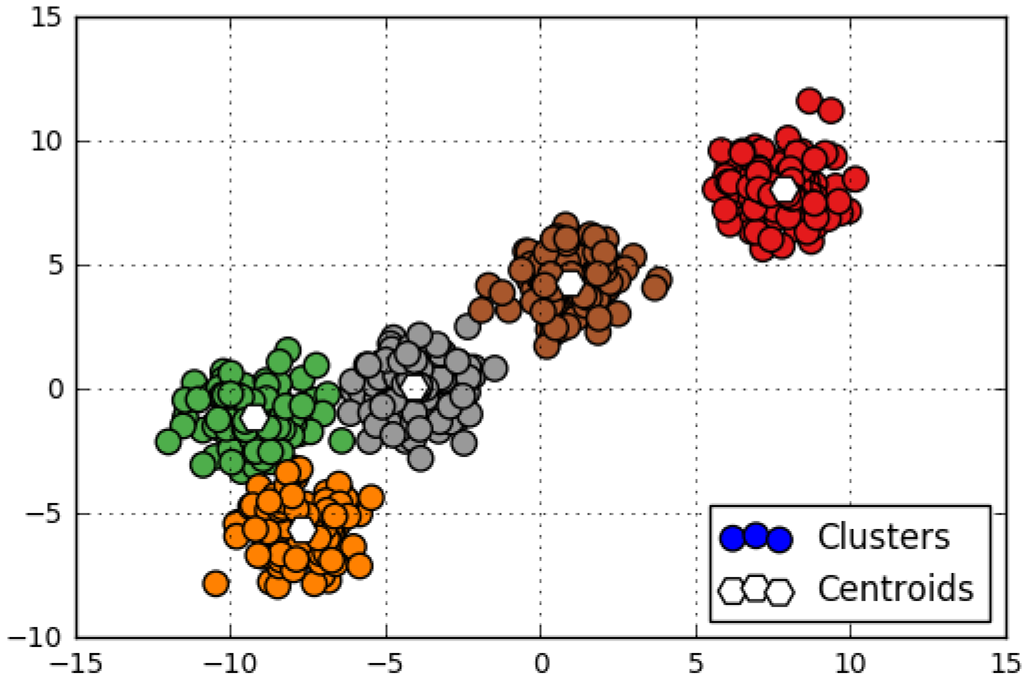
```
centroids = KM_model.cluster_centers_
KM_model.cluster_centers_
```

```
array([[ 7.82245556,  8.06663956],
       [-9.23774973, -1.08430495],
       [-7.67898131, -5.60470315],
       [ 0.9387585 ,  4.29843952],
       [-4.09690757,  0.13966778]])
```

```
clusters = KM_model.labels_
KM_model.labels_
```

```
array([1, 0, 4, 4, 0, 3, 1, 1, 1, 2, 1, 1, 4, 4, 4, 2, 0, 2, 1, 0, 3, 0, 1,
       3, 0, 4, 2, 1, 3, 1, 1, 3, 4, 3, 2, 0, 3, 4, 3, 4, 0, 2, 2, 4, 1, 3,
       1, 3, 1, 0, 4, 0, 3, 4, 0, 3, 0, 1, 1, 4, 1, 2, 2, 0, 3, 2, 1, 0, 3,
       4, 1, 3, 4, 1, 0, 3, 2, 2, 0, 2, 1, 2, 0, 4, 1, 3, 0, 2, 1, 2, 0, 3,
#Let's plot the clusters generated by the model
fig, ax = plt.subplots()
ax.scatter(x[:,0],x[:,1], marker='o',c=clusters, s=80,
           cmap='Set1', label ='Clusters')
ax.scatter(centroids[:,0],centroids[:,1],marker='H', s=110,
           c='white', label='Centroids' )
plt.grid()
plt.legend(loc=4,)
```

```
<matplotlib.legend.Legend at 0x1af49d25a58>
```

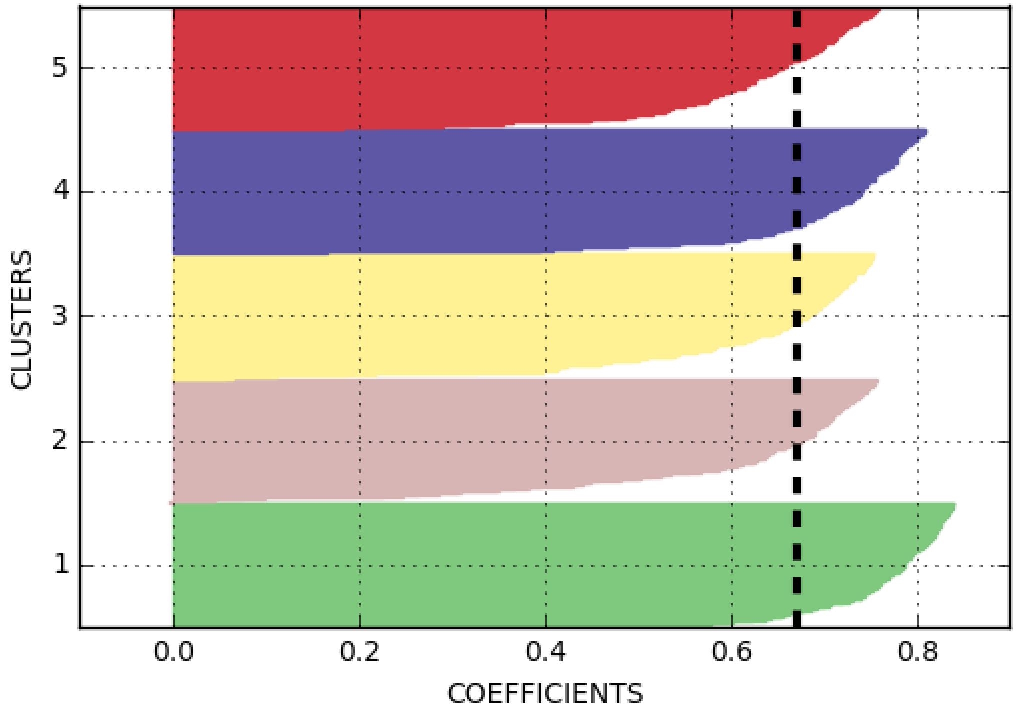


```
from sklearn.metrics import silhouette_samples

# to find the silhouette coefficient of every observation
#(measured by euclidean distance by default)
sil_coef=silhouette_samples(x,clusters)
sil_coef
```

```
array([ 0.69447886,  0.83048724,  0.62567642,  0.3549017 ,  0.8341171 ,
        0.67926329,  0.69162779,  0.44169701,  0.22669595,  0.4614391 ,
        0.45840283,  0.63500886,  0.65716309,  0.72952993,  0.66507429,
        0.67534136,  0.82497281,  0.4602552 ,  0.74671046,  0.8298624 ,
        0.71332668,  0.82599163,  0.73056877,  0.79525011,  0.81970064,
        0.60274579,  0.21785004,  0.69991922,  0.77111636,  0.67174625,
        0.60317144,  0.74141965,  0.64835867,  0.55052813,  0.63043696,
```

-SILHOUETTE PLOT- ¶

```
# to set the labels (0 to 4)
list_labels=np.unique(clusters)

# to set number of clusters (5)
clusters_total=list_labels.shape[0]

y_low,y_up=0,0 #upper and lower boundaries for bars (barplot)
ticks =[]

#to plot the silhouettes, 1 by 1 through iteration
for a,b in enumerate(list_labels):

    #to create a group of coefficients per each cluster label
    val_c = sil_coef[clusters==b]

    #to sort the coefficients in the group
    val_c.sort()

    #to increase the number of values of y axis after every iteration
    y_up += len(val_c)

    #to generate different colors after each iteration
    cl =cm.Accent(X=a/clusters_total)

    #to plot the cluster (horizontal bar graph)
    plt.barh(bottom=range(y_low, y_up),
             width=val_c,
             height=1.0,
            edgecolor=cl)

    ticks.append((y_low+y_up)/2)

    #to raise the lower boundary for next iteration
    y_low += len(val_c)

#-------------------------------------------------------
#to get the coefficients average and plot it as a line
coef_avg = np.mean(sil_coef)
plt.axvline(coef_avg, color="black", ls='--', lw=3)

#to modify the ticks
plt.yticks(ticks, list_labels+1)

plt.ylabel('CLUSTERS')
plt.xlabel('COEFFICIENTS')
plt.grid()
```

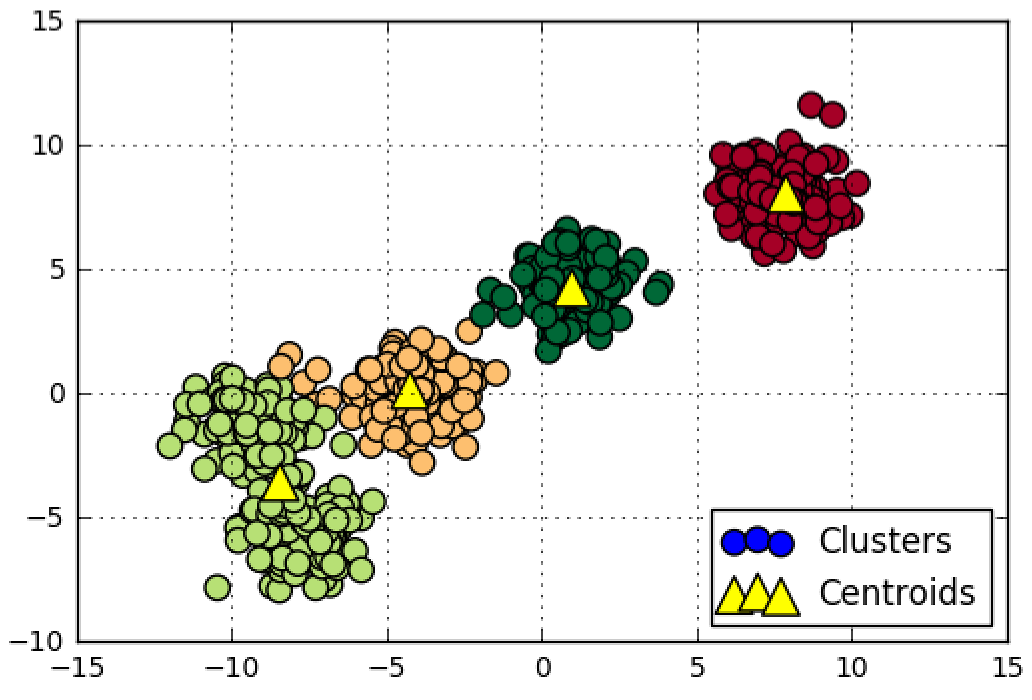


This silhouette plot shows a strong cohesion within each cluster. Each cluster, represented by each color, is the aggregation of the coefficients of every data point within its group. All clusters are very far from 0 and well above the average (.67). Since we knew beforehand the correct means we could achieve these results. Let's plot another silhouette graph with a different (and erroneous) value for k:

```
#Let's create the model with k=4
KM_model = KMeans(n_clusters=4, init='random', random_state=0)
KM_model.fit(x)
centroids = KM_model.cluster_centers_
clusters = KM_model.labels_
```

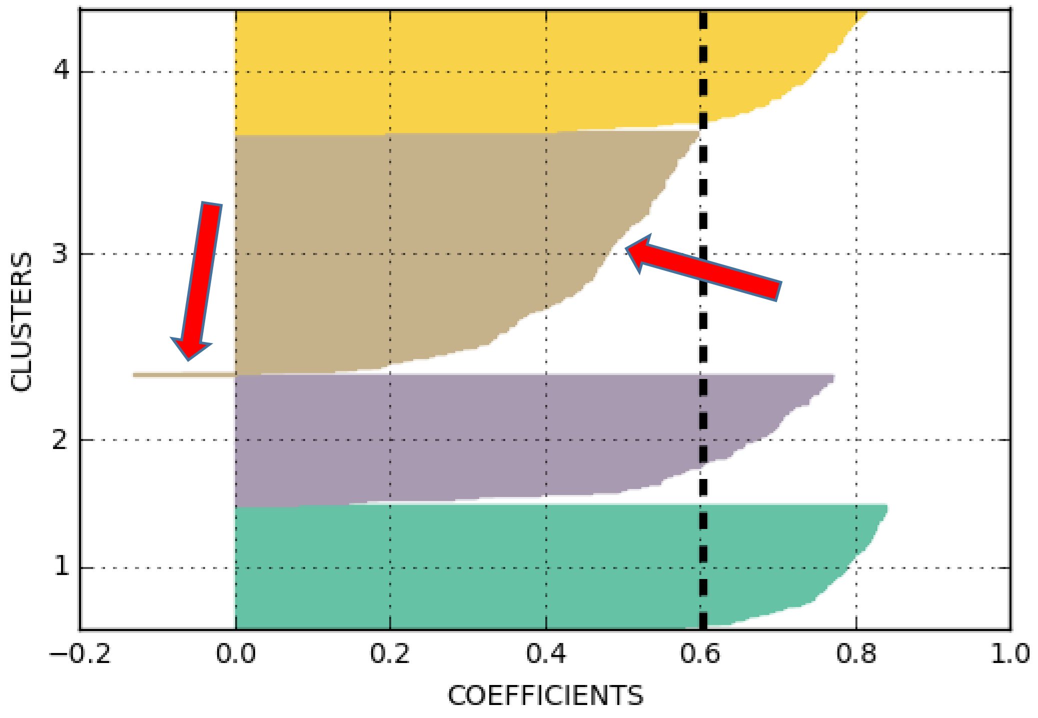
```
#Let's plot the clusters generated by the model
fig, ax = plt.subplots()
ax.scatter(x[:,0],x[:,1], marker='o',c=clusters, s=80,
           cmap='RdYlGn', label ='Clusters')
ax.scatter(centroids[:,0],centroids[:,1],marker='^', s=180,
           c='yellow', label='Centroids' )
plt.grid()
plt.legend(loc=4,)
```

```
<matplotlib.legend.Legend at 0x2656137db00>
```



```
#Let's generate the coefficients
sil_coef=silhouette_samples(x,clusters)
```

```
#Let's plot the results
list_labels=np.unique(clusters)
clusters_total=list_labels.shape[0]
y_low,y_up=0,0
ticks =[]

for a,b in enumerate(list_labels):

    val_c = sil_coef[clusters==b]
    val_c.sort()
    y_up += len(val_c)
    cl =cm.Set2(X=a/clusters_total)
    plt.barh(bottom=range(y_low, y_up),
             width=val_c,
             height=1.0,
            edgecolor=cl)
    ticks.append((y_low+y_up)/2)
    y_low += len(val_c)

coef_avg = np.mean(sil_coef)
plt.axvline(coef_avg, color="black", ls='--', lw=3)
plt.yticks(ticks, list_labels+1)

plt.ylabel('CLUSTERS')
plt.xlabel('COEFFICIENTS')
plt.grid()
```



```
sil_coef[sil_coef<0]
```

```
array([-0.13044773, -0.06842284])
```

The plot shows a week clustering due to the brown section being under the mean line. This indicates poor cohesion within the cluster and asks to review k. There is also a couple of coefficients below 0 which indicates a misclassification of those 2 data points.

These 2 methods for cluster selection (elbow) and quality test (silhouette) are only the starting point to produce a well-crafted clustering model. There are many considerations that I haven't covered but are essential. For example, we should consider the possibility of outliers in our dataset that might distort our cluster results. Or, once we have a cluster, is it the representation of a real group or is it only appearing due to mere randomness? To address these issues, other techniques must be used. Generally, what we want is to assign probabilities instead of a single label; this is known as *soft clustering.* This category has its own algorithms and applications, but it is extremely long to be covered here. This documents' main goal has been to present a general overview of clustering and unsupervised learning.

References

James, Gareth. *An Introduction to Statistical Learning*. New York: Springer. 2013. Print.

Wikipedia contributors. "k-means clustering." *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 22 Nov. 2016. Web. 06 Dec. 2016. < https://en.wikipedia.org/wiki/K-means_clustering>

Scikit-learn. “Selecting the Number of Clusters with Silhouette Analysis on KMeans Clustering.” *scikit-learn.org*. 06 Dec. 2016. < http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html>

Raschka, Sebastian. *Python Machine Learning*. Birmingham: Packt Publishing. 2015. Print.