Assignment 6: Unsupervised Clustering - Machine Learning 6316

Jacob Dineen / JD5ED

November 29, 2019

1 KMeans

This will be a relatively short write-up, as this was said to be extra credit only. I wasn't exactly sure about the flow of the existing template, so I made some modifications to adapt to the way that I've learned about K-means in the past, trying to keep the existing method structure.

- Initialize Random Centroids.
- For each datapoint, get the distance between it and the cluster centroids.
- Choose the argmin distance as that data points assigned cluster.
- Update Centroids. Start with the randomly initialized ones. At each iteration, calculate the mean of the datapoints that belong to each cluster. These will act as the new cluster centroids.
- Repeat for t iterations, or until the between cluster variance and within cluster variance is minimized. We used a finite length of time here, although convergence appears to happen quickly.

Figure 1, noted on page 2, shows the various clusters generated by k = 1:6. This was further validated by checking against scikitlearn. The results are very close. To generate these results, the following methods were used:

- load data, of course
- randomly select initial centroids
- Allocate points based on initial centroids
- updateCentroids according to mean values of features within each cluster.
- Iterate using the main kmeans method.
- visualize results for various k.

Each datapoint is colored according to the classification of their belonging cluster, as determined by the min distance between that data point and the closest cluster. I've also plotted the cluster centroids superposed on top of the scatter plots for both the hand-crafted code and sklearn implementation. I let the 'from-scratch' methods run for 300 epochs.

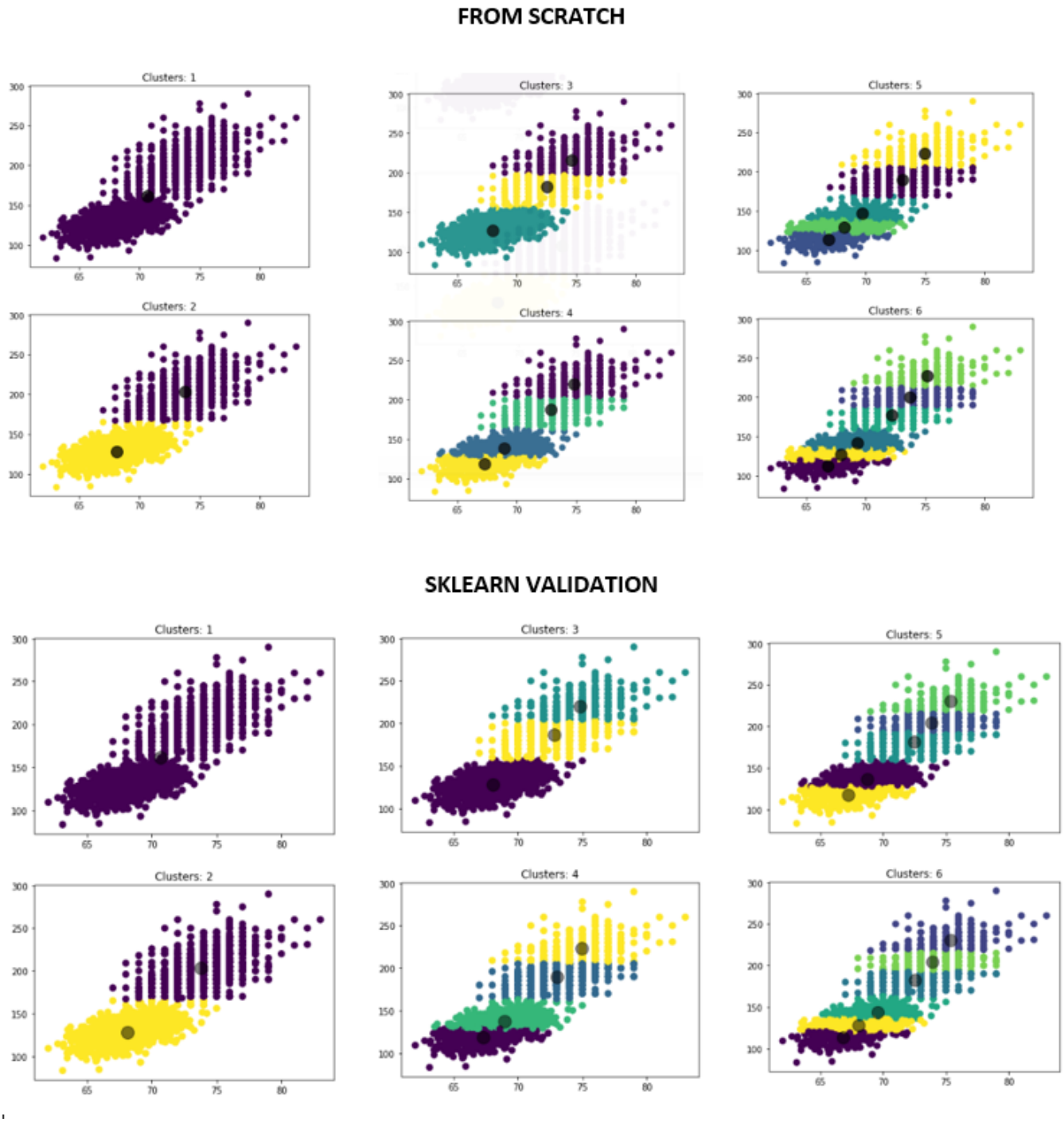


Figure 1: Showing Clusters w/ various k

A look at the elbow plot below shows the total sum of square errors between each element in a cluster against the cluster centroid position for various values of k, as per above. We run this from between 1 and 6 clusters. There is a large drop-off in total error going from one cluster to 2, and then some slight decay afterwards, i.e. there's not much benefit in increasing the size of k past 2 (especially past 3). I hadn't previously used the purity metric, but

```
{1: 4249845.887735559,
 2: 678810.937568983,
 3: 374668.98067819723,
 4: 248847.82267064592,
 5: 238584.28595261966,
 6: 128792.78617529431}
```

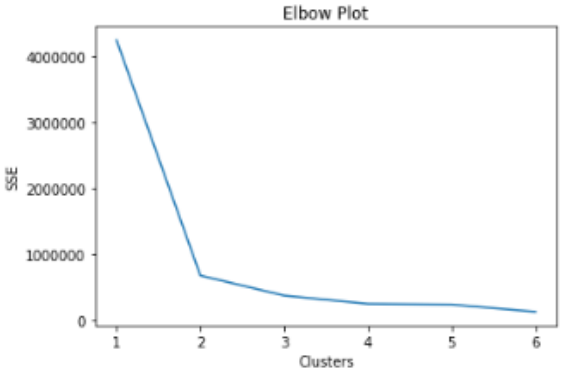


Figure 2: Total SSE given various clusters

after some research/stackoverflow, it appears to be a goodness of the clustering assignments compared against the actual class labels. A higher purity, approaching one, means that we successfully clustered the data per the classes. I had to make some minor tweaks here. Namely, the true class labels were 1 and 2, so I had to subtract one. Second, the clusters assigned to the datapoints were flipped (because that doesn't really matter when clustering) so I had to use some numpy logic to flips 0's to 1's and vice versa. See screenshot below for the purity of the clustering classification with k = 2 and maxiter set to 500. Figure 4 shows the confusion matrix between ytrue and ypred. To find the purity, we sum up the diagonal and divide it by the total sum of the matrix, similar to how we would for computing accuracy from a confusion matrix. The purity when k=2 is ¿98 percent, speaking to high congruence between our clustering classifications and true labels.

```
print('Purity:', purity(y = dataset1_y, clusters= clusters_assignments))
```

executed in 6ms, finished 21:18:36 2019-11-21

```
Purity: 0.9843330349149507
```

Figure 3: Showing Clusters w/ various k

```
[[1.199e+03 1.000e+00]
 [3.400e+01 1.000e+03]]
```

Figure 4: Showing confusion matrix

This part of main will output: cluster SSE for kneefinding plot (each of 6 clusters). Elbow plot. Purity of k = 2 kmeans clustering and an output scatter plot showing the two cluster centroids plotted ontop of the original data.

2 Gaussian Mixture Models

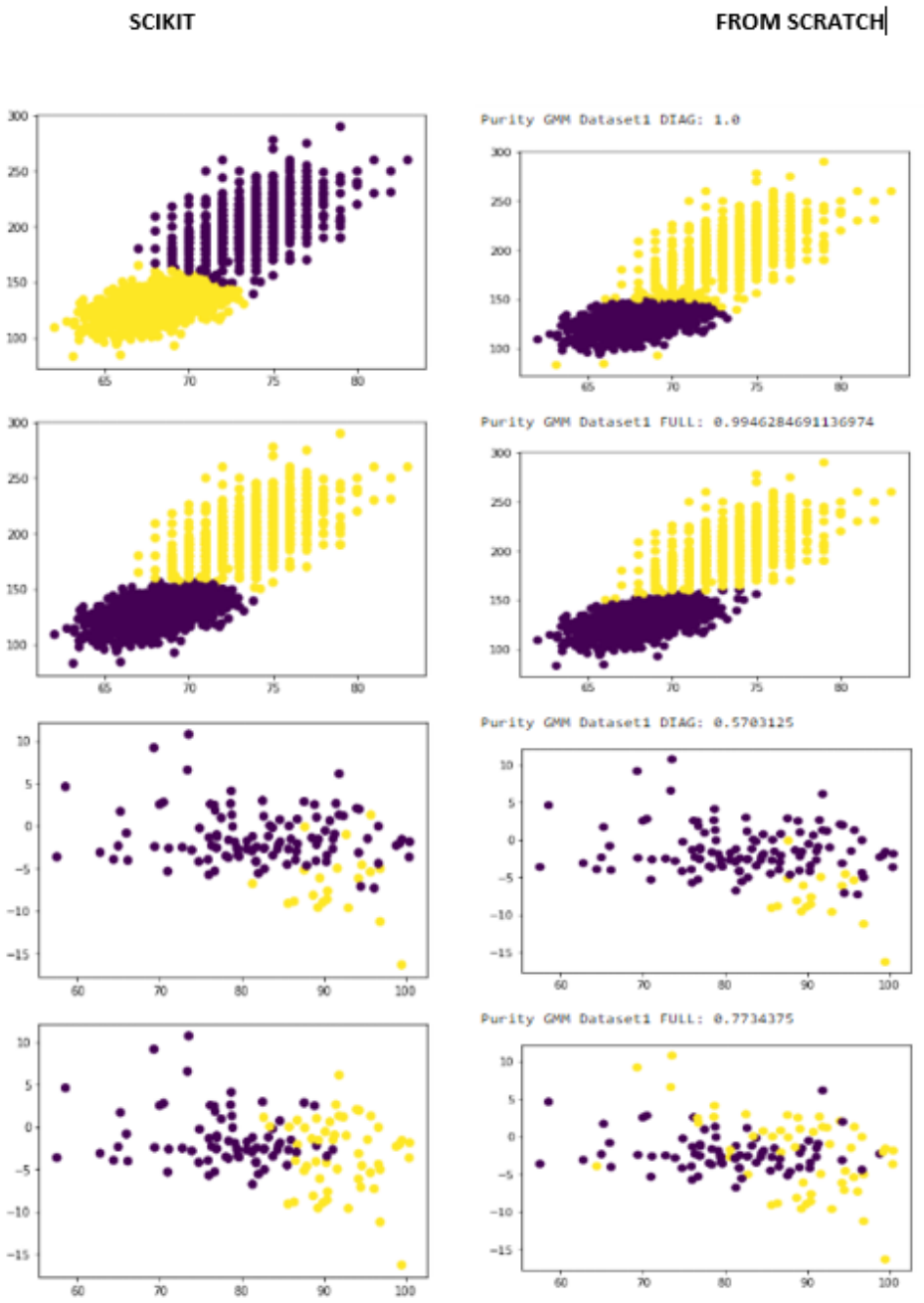


Figure 5: GMM scatter plots: Running each dataset through GMM method plotting cluster assignment along with first two columns of both datasets. Hand coded versions show purity metrics to compare cluster assignments to true class labels

Note - I rewrote a GMM code tutorial to work here using the initial covariance matrix code that was started for us. There were some issues with the original template, mainly - We have to update the covariance matrices in the M step, so having them bounded within the initial method didn't make sense to me. Second, the purity metric should

be the same for both Kmeans and GMM, as it's just showing the relationship between cluster assignments and true class labels.

The flow of the process, as I understand it, is as follows:

- Start with some random covariance matrix: we start with the true covariance matrix here (diag or full) and clone it because each cluster will have its own cov matrix to determine the shape of the sphere of elipsis. Our cov var is a p x p x k array.
- Start with some random means. This is feature wise, so we just randomly select k samples from the data. This is a k x p array.
- Start off with some random mixing proportion. We uniformly assign this as a 1xk vector, where each element is a uniform probability according to k. We need to learn the true proportion of samples that fall into each cluster/class. This comes into play during the M step.
- We calculate the densities of each datapoint using a multivariate normal distribution and each clusters mean and covariance arrays.
- On the E Step, we assign (softly) responsibilities/probabilites of each datapoints assignment to each cluster. this is an n x k array. We eventually take the argmax of this to determine cluster assignment.
- In the M step, we update our covariances, means and mixing probabilities.
- This process is iteratively stepped through.

3 Sample QA Questions:

Question: 2. Decision Trees The following dataset will be used to learn a decision tree for predicting whether a person is happy (H) or sad (S) based on the color of their shoes, whether they wear a wig and the number of ears they have.

(a) [2 points] What is $H(Emotion|Wig = Y)$ (where H is entropy)? Using log base 2.

$P(Emotion = H|Wig = Y)*log(P(Emotion = H|Wig = Y))+P(Emotion = S|Wig = Y)*log(P(Emotion = S|Wig = Y))$

$$H(Emotion|Wig = Y) = -(1/2log1/2 + 1/2log1/2)$$

=-(-.5 + -.5) = 1

(b)[2 points] What is H(Emotion — Ears = 3)? Using log base 2.

$$H(Emotion|Ears = 3)$$

$P(Emotion = H|Ears = 3)*log(P(Emotion = H|Ears = 3))+P(Emotion = S|Ears = 3)*log(P(Emotion = S|Ears = 3))$

-(1 * log 1 + 0 * log 0) = 0

(c) [3 points] Which attribute would the decision-tree building algorithm choose to use for the root of the tree (assume no pruning)?
You'd use color, as it provides the highest information gain. See work below - Figured I needed all of these computations to build the entire tree anyway

$$H(Y) = -(4/9log4/9 + 5/9log5/9) = 0.99$$

Starting with X1 (Color):

$$H(Y|Color) = P(Color = G)H(Y|Color = G) + P(Color = B)H(Y|Color = B) + P(Color = R)H(Y|Color = R)$$

$P(Color = G) = 3/8$
$P(Color = B) = 2/8$
$P(Color = R) = 3/8$
$H(Y|Color = G) = -(1log1 + 0log0) = 0$
$H(Y|Color = B) = -(1/2log1/2 + 1/2log1/2) = 1$
$H(Y|Color = R) = -(1log1 + 0log0) = 0$

$$H(Y|Color) = 3/8 * 0 + 2/8 * 1 + 3/8 * 0 = 0.25$$

IG(Color, Y) = 0.99 - 0.25 = 0.74

X2 (Wig):

$$H(Y|Wig) = P(Wig = True)H(Y|Wig = True) + P(Wig = False)H(Y|Wig = False)$$

$P(Wig = True) = 7/9$ $P(Wig = False) = 2/9$ $H(Y|Wig = True) = -(1/2log1/2 + 1/2log1/2) = 1$
$H(Y|Wig = False) = -(3/7log3/7 + 4/7log4/7) = 0.985$

$$H(Y|Wig) = 7/9 * 1 + 2/9 * 0.985 = 0.99$$

IG(Wig, Y) = 0.99 - 0.99 = 0

Onto X3 (Num Ears):

$$H(Y|NumEars) = P(NumEars = 2)H(Y|NumEars = 2) + P(NumEars = 3)H(Y|NumEars = 3)$$

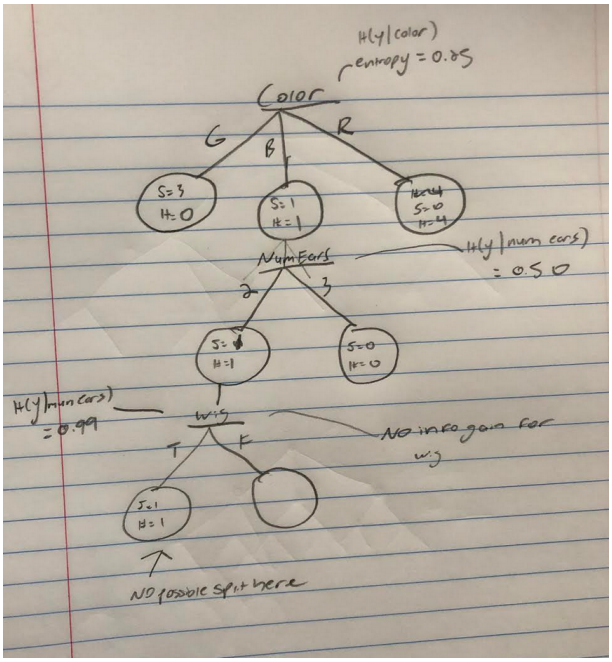
$P(NumEars = 2) = 8/9$
$P(NumEars = 3) = 1/9$

$H(Y|NumEars = 2) = -(8/9log8/9 + 1/9log1/9) = 0.50$
$H(Y|NumEars = 3) = -(1/9log1/9 + 8/9log8/9) = 0.50$

$$H(Y|NumEars) = 8/9 * 0.5 + 1/9 * 0.5 = 0.5$$

IG(Num Ears, Y) = 0.99 - 0.50 = 0.49

(d) [3 points] Draw the full decision tree that would be learned from this data (assume no pruning). Because We don't gain anything from splitting on wig, we could limit this to splitting on the two features with IG ¿ 0. Please find my hand-drawn image below. There is no possible way to split that would result in 100 percent accuracy given the current feature split. Observations 4 and 5 have the same independent feature, but different responses. Splitting on color, however, gives us two pure nodes right off the bat.



(e) [3 points] Assuming that the output attribute can take two values (i.e. has arity 2) what is the maximum training set error (expressed as a percentage) that any dataset could possibly have? **In occurrences where we can't get a pure split - 50 percent error on the training split. This would happen if pairwise inputs were exactly similar, but had different labels. See below for dataset construction**

(f) [3 points] Construct an example dataset that achieves this maximum percentage training set error (it must have two or fewer inputs and five or fewer records).
In the case below, the information gain due to the uniformity of both the independent and dependent features would be 0. There is no possible way to split the data here.

Table 1: Toy Dataset

X1	X2	Y
0	1	0
0	1	1
1	0	0
1	0	1