CS5350: Machine Learning *Due end of semester*

Project 5: Clustering

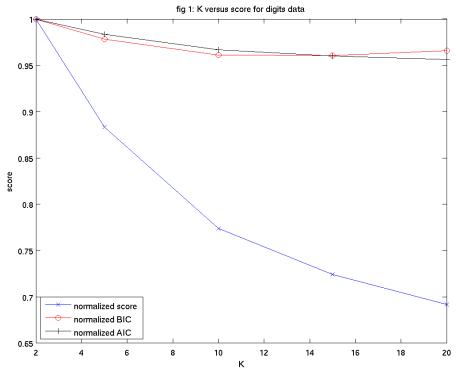
Putting it all together... *(30%)*

As usual, there is a `run` script that puts this all together. This one generates a *large* number of plots. However, in contrast to previous projects, it only takes about few minutes to run to completion. All of this is based on the digits data from P1.

Please answer the questions associated with the figures in your write-up. This produces a fairly large number of figures, but doesn't take too long (under 10 minutes for me on my crummy laptop).

1. For experiment 1, we just do K-means clustering with the furthest-first initialization on the MNIST data with different values of K. The scores you get should be roughly $22600, 19900, 17500, 16400, 15600$. This takes me about $1.5 minutes$. It produces the following Figures:

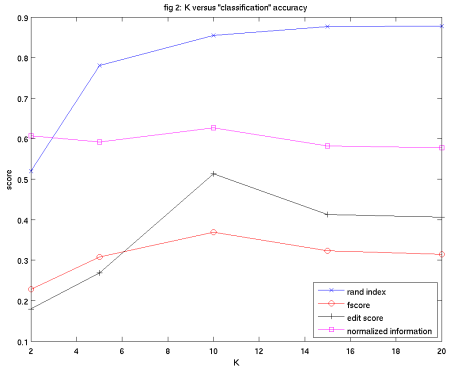
   Fig 1 displays a plot of the *scores* produced by K means (the blue line) as well as "regularized" scores that try to take into account whether having more clusters is "worth it" or not. The two versions of regularization we use are called BIC (Bayes Information Criteria) and AIC (Akaike Information Criteria). If you had to choose a value of K using each of the three scoring methods, which would you choose? Based on what you know about this data, which scoring function do you like best?

   Wei: (if We assume higher scores are better...) by only K-means, one would choose $K = 2$, which is not true if we know the mnist data has ten classes actually. If we use BIC or AIC, it's hard to choose because all K have similar scores. I'd like to see $K = 10$ have higher scores, but actually its score is even lower than $K = 2, 5$. But with regularized scores, at least $K = 2$ is not much better than othes.

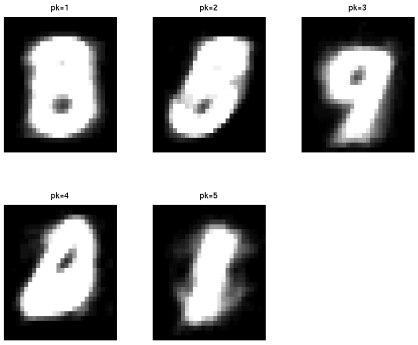
   If I have to choose, AIC is best among them, as it has highest scores at $K = 10$.

   Fig 2 displays "classification" accuracy results for the different values of K. That is, we compare the clusters produced by kmeans to the *true* clusters. In all cases, higher scores are better. The different lines are different methods for computing such a "similarity of clusters" score. Which seem to have good behaviour? Why?

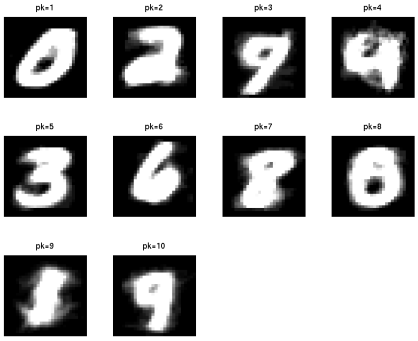


edit score is best, because it has a highest scores at $K = 10$

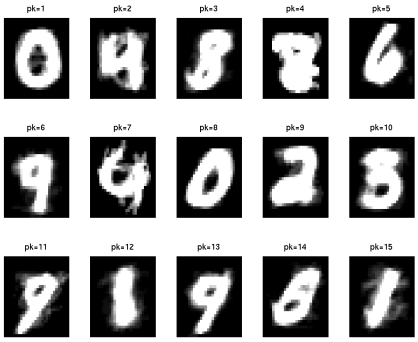
Fig 3-5 display the means found for $K = 5, 10, 15$. For each of these, do you see clusters that look like the actual digits? How big does K have to be before you essentially see a mean for each "true" digit? Why do you suppose some digits are "over-represented?"



I could not see the digits from the means, as the $K = 5$ is less than the true number of classes.



I can tell most digites from this figure, except number 5, 7 and 8.

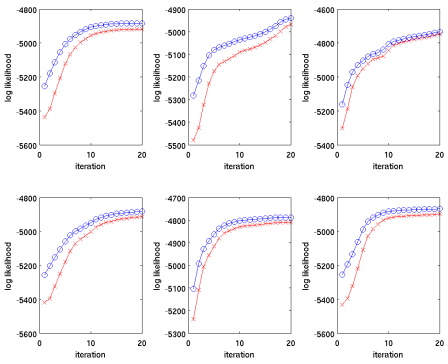


This 'over-represented' means have more than one '9', '0' and '8'.

So we can see $K = 10$ is really the least number of classes that I can see a digit from the mean image. Some digites are 'over-represented' because I can see two means actually is same digits.

2. For experiment 2, we first reduce the dimensionality of the data using PCA. We then test the Gaussian mixture model implementation for 5 classes. This takes me 30 seconds.

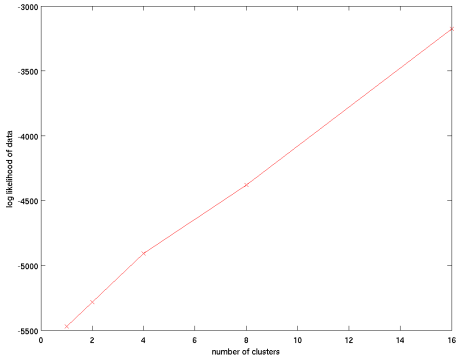
   Fig 6 plots the complete- (red xs) and incomplete- (blue os) log likelihoods for six runs of GMM with five clusters. As a verification of implementation, you should see that the *incomplete* log likelihood is always increasing. Moreover, the complete should lower-bound the incomplete (i.e., the blue line should always be above the red line). Is there a significant difference between the ILLs and CLLs? Do the CLLs seem to converge roughly to the same value across all runs?



There is not much difference between the ILLs and CLLs. CLLs did converge. Among 6 figures, 4 figures have roughly same convergence values for CLLs, which is not bad.

3. Next, the GMM is tried with different number of clusters, from 1 to 16. This takes me 5.5 minutes.

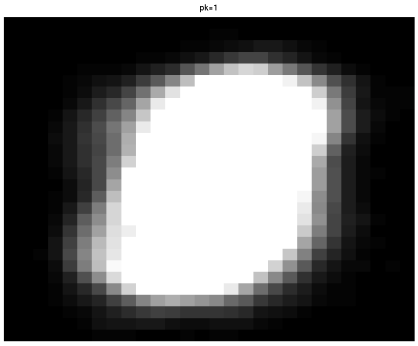
   Fig 7 shows the incomplete log likelihood as a function of number of clusters. It should be more or less monotonically increasing. Why doesn't it drop after $K = 10$, which seems to be a "good" number of clusters for this data?



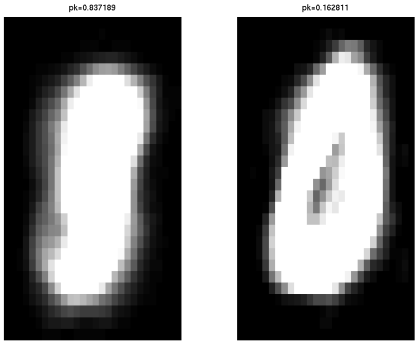
Because it is not regularized. If we add some term to verify if it benefits to increase number of clusters (just like AIC or BIC in K-mean), we may get a drop after $K = 10$.

Another explanation is, suppose number of classes increases so much as to equals the number of points, then theoretically we know $p(z|x)$ given x, so the complete log likelihood is the largest. And accordingly the ILL is also the largest. This is consistent with this figure.

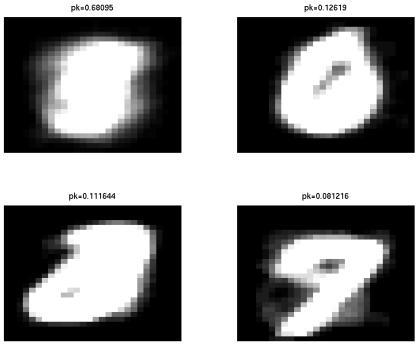
Fig 8-12 show the means found by GMM for different values of K. How do these compare to the K-means means? How big does K have to be before you essentially see a mean for each "true" digit? Why do you suppose some digits are "over-represented?" Does this differ from your answer to the (nearly identical) question about K-means?



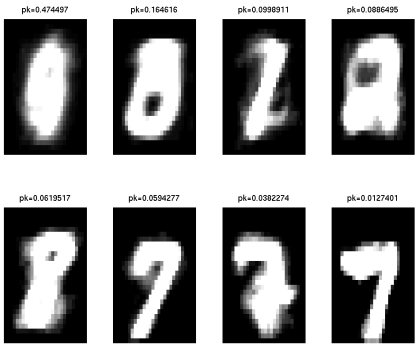
They are similar to K-means means. That is, when number of classes is too small, GMM means can not represent the true center well.



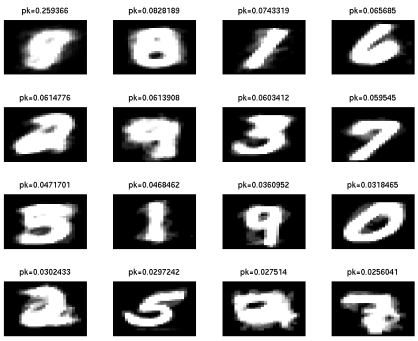
$K = 2$ is not enough to figure out what digits are in these means images.



$K = 4$ is not enough to figure out what digits are in these means images.



I can tell half the digits in the images above.

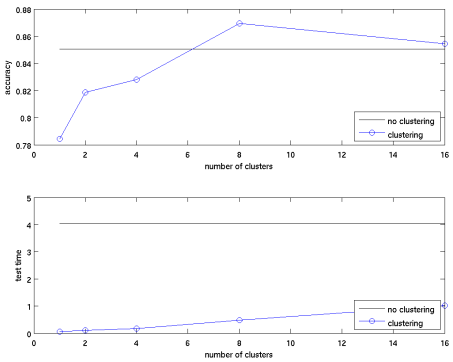


This one is really too many K. We can find two digits '1', and two digit '7' in them.

4. Finally, we use clustering as a method of speeding up K nearest neighbor classification. What we do is: for each class y, generate K clusters. We think of these K clusters as "prototypical" points for class y. Thus, for MNIST, since there are 10 classes, we end up with $10 \times K$ points. These are now the "training data" that is fed into the KNN classifier. When $K = 1$, each class just gets a single representative; when $K = 5$, each class gets five representatives. This takes about 10 seconds for me.

   Fig 13 displays test accuracy and test time as a function of the number of clusters per (true) class. The test time should monotonically increase as a function of the number of clusters (per class). Does test time increase linearly? Why or why not? Does the test accuracy monotonically increase:

that is, does adding more clusters always help? Why or why not? What number of clusters would you choose?



The test time increase linearly. This is because in test phase, we need compute the distance of the test point and each points in $10K$ training points, so the computation cost is $\mathcal{O}(K)$.

The accuracy does not increase monotonically. This is because, if we have more clasters, we tens to introduce more noise. And if the test point happen to be close to the noise points, it will be incorrectly classified.