CS273a Homework #5

Introduction to Machine Learning: Fall 2016

Due: Friday December 2nd, 2016

Write neatly (or type) and show all your work!

Please remember to turn in at most two documents, one with any handwritten solutions, and one PDF file with any electronic solutions.

Problem 1: Basics of Clustering

The code this week provides the three clustering algorithms we discussed: k-means, agglomerative clustering, and EM for Gaussian mixture models; we will explore the first two here. (These functions are also provided in many 3rd party toolboxes; you are free to use those if you prefer.) In this problem, you'll do some basic exploration of the clustering techniques.

(a) Load the usual Iris data restricted to the first two features, and ignore the class / target variable. Plot the data and see for yourself how "clustered" you think it looks. (You don't have to report on this.)

(b) Run k-means on the data, for $k = 5$ and $k = 20$. For each, turn in a plot with the data, colored by assignment, and the cluster centers. (You can easily do this yourself manually, using `ml.plotClassify2D(None,X,z)`, where z are the resulting cluster assignments of the data.) Try a few different initializations and check to see whether they find the same solution; if not, pick the one with the best score.

(c) Run agglomerative clustering on the data, using *single linkage* and then again using *complete linkage*, each with 5 and then 20 clusters. Again, plot with color the final assignment of the clusters, and describe their similarities and differences from each other and k-means. (This algorithm has no initialization issues; so you do not have to try multiple initializations.)

(d) **(Optional)** Run the EM Gaussian mixture model with 5 components. (Note: if you also decide to try 20 components, you may get some rank deficiency errors.) As with k-means, you may want to try several initializations. Again, compare / discuss differences with the other clusterings. Which do you think is most reasonable?

As a side note: Clustering is often a useful element of other predictive tasks, like supervised learning. To be used properly, you need to be able to define the "out of sample" cluster assignments, but this is very easy for k-means and EM (a bit less so for agglomerative); for k-means, say:
`crule = ml.knnClassify( clusters, np.arange(k), 1 ); z = crule.predict( X );`
Then, you can then use these cluster assignments as a feature in a classifier:
`Phi = lambda x:  ml.to1ofK( crule.predict(x) , np.arange(k) );` will create k new binary features indicating which of the clusters is closest to a new point x.

Problem 2: EigenFaces

In class I mentioned that PCA has been applied to faces, and showed some of the results. Here, you'll explore this representation yourself. First, load the data and display a few faces to make sure you understand the data format:

```
X = np.genfromtxt("data/faces.txt", delimiter=None)  # load face dataset
plt.figure()
img = np.reshape(X[i,:],(24,24))                     # convert vectorized data point to 24x24 image pat
plt.imshow( img.T , cmap="gray")                     # display image patch; you may have to squint
```

(a) Subtract the mean of the face images ($X_0 = X - \mu$) to make your data zero-mean. (The mean should be of the same dimension as a face, 576 pixels.)

(b) Use `scipy.linalg.svd` to take the SVD of the data, so that

$$X_0 = U \cdot \mathrm{diag}(S) \cdot Vh$$

Note that since the number of data is larger than the number of dimensions, there are at most 576 non-zero singular values; you can use `full_matrices=False` to avoid using a lot of memory. Like the slides, I suggest computing `W = U.dot( np.diag(S) )` so that $X_0 \approx W \cdot Vh$.

(c) For $K = 1 \ldots 10$, compute the approximation to X_0 given by the first K eigendirections, e.g., $\hat{X}_0 = W[:, :K] \cdot Vh[:K, :]$, and use them to compute the mean squared error in the SVD's approximation, `np.mean( (`$X_0 - \hat{X}_0$`)**2 )`. Plot these MSE values as a function of K.

(d) Display the first three principal directions of the data, by computing $\mu + \alpha$ V[j,:] and $\mu - \alpha$ V[j,:], where α is a scale factor (I suggest, for example, `2*np.median(np.abs(W[:,j]))`, to get a sense of the scale found in the data). These should be vectors of length $24^2 = 576$, so you can reshape them and view them as "face images" just like the original data. They should be similar to the images in lecture.

(e) Choose two faces and reconstruct them using only the first K principal directions, for $K = 5, 10, 50$.

(f) Methods like PCA are often called "latent space" methods, as the coefficients can be interpreted as a new geometric space in which the data are being described. To visualize this, choose a few faces at random (say, about 15–25), and display them as images with the coordinates given by their coefficients on the first two principal components:

```
idx = ...      # pick some data at random or otherwise; get list / vector of integer indices

import mltools.transforms
coord,params = ml.transforms.rescale( W[:,0:2] )  # normalize scale of "W" locations
for i in idx:
    loc = (coord[i,0],coord[i,0]+0.5, coord[i,1],coord[i,1]+0.5)  # where to place the image & siz
    img = np.reshape( X[i,:], (24,24) )
    plt.imshow( img.T , cmap="gray", extent=loc ) # draw each image
    plt.axis( (-2,2,-2,2) )                       # set axis to reasonable visual scale
```

This can often help you get a "feel" for what the latent representation is capturing.

Problem 3: Go work on your project