CS 4641: Machine Learning
Homework 3 — Unsupervised Learning

Instructions

Submission

Please submit **one** zipfile containing a folder with **all** your code, data files, and a **single** pdf with your answers to the written questions. This zipfile should have the following format: `hw3-GTusername.zip`, replacing "GTusername" with your username (for example "hw3-bhrolenok3.zip").

For this assignment, your submission should include:

- `hw3-answers.pdf` A PDF of your answers to the written questions.
- `hw3.py` The template file, modified to include your code.
- `README` Any special instructions for running your code, (library versions, network access, etc) **plus** any external sources you used to help you with this assignment. This includes other students and websites. When in doubt, cite it!

Collaboration

All of the assignments in this class are **individual work only**. You're welcome to *discuss* the math or high level concepts behind the algorithms you're being asked to implement, but you are **not allowed to share code or answers to the written questions**. We will be checking, manually and with automated tools, for any violations of this policy, so please **do not share solutions or code**.

Debugging tips

In order to verify that your learners are working correctly, you can compare your results with those from a library such as `scikit-learn`[1]. For a quick debugging check, running the `main` method in the skeleton file should give you results for a simple debugging dataset that has only two dimensions, and should be easy to visualize. Figure 1 shows the expected output for working PCA and k-Means implementations. The components found by PCA should be $[0.66, -0.75]^\top$ and $[0.75, 0.66]^\top$ (sign flips are OK as long as the vectors are orthogonal). The cluster centers found by k-Means should be near $[-1.3, -1.7]^\top$ and $[1.5, 1.5]^\top$. The reconstruction error when the number of components is the same as the original dimensionality should be close to zero for any dataset; for the test data the reconstruction error for 2 components is 3.1×10^{-15}.

Principal Component Analysis

PCA is a very popular feature transformation technique that has nice theoretical properties and is still surprisingly simple to implement. The central method is based on the Singular Value Decomposition, which states that any matrix can be re-written as the product of three different matrices with useful properties:

$$X = USV^\top$$

[1] `https://scikit-learn.org/`

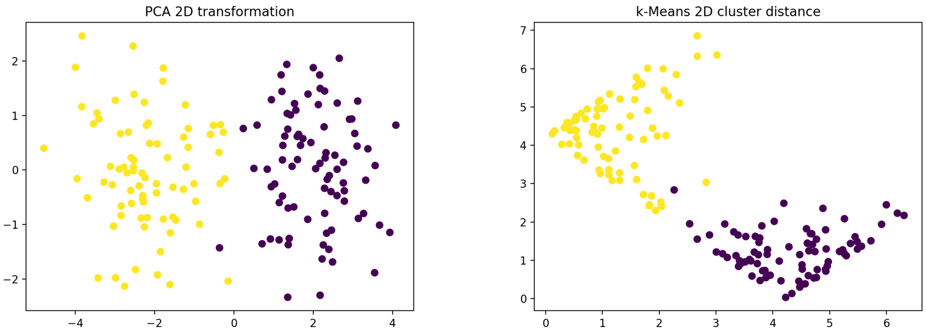


Figure 1: Expected output for PCA and k-Means on the 2D test dataset. The Normalized Mutual Information score should be close to 0.95.

For our purposes, the V matrix is the most interesting, because it contains as its columns the eigenvectors of the covariance matrix of X, which as we showed in class, are exactly the vectors that point in the direction of "highest variance." To actually solve for this decomposition, we can use the Numpy library, specifically the function `numpy.linalg.svd`. **Note 1**: read the documentation for this function carefully, so that you know what you need to pass in and what is being returned. Use this function to finish the implementation of `PCA.find_components`, which should simply store the return values in the appropriate data members. **Note 2**: don't forget to center the data (subtract out the mean) before passing it to the `svd` function, otherwise your results may be different.

With the V matrix in hand, computing the actual feature transform is very simple. If X is the centered data, and V_L is the matrix consisting of the first L columns of V, we can compute the transformed data X' as follows:

$$X' = XV_L$$

Use this equation to implement the `PCA.transform` method, making sure to center the data, and only use the first L columns[2] of V. For the final method you need to implement, we can leverage one of the special properties of V, namely that $V^\top = V^{-1}$. This means we can write the *inverse* transform as:

$$X'' = X'V_L^\top$$

Implement this in `PCA.inv_transform` to complete the class, making sure to "undo" the centering you did in the original transform (add the mean back in)[3]. With all three of these methods finished, your PCA implementation should be complete, and you can test out the plotting methods provided for you to visualize any high dimensional dataset by projecting it down to the first two or three principal components. You can also see how much "information" is lost in these projections by looking at the reconstruction error, which is the difference between the original data, and the data passed through a low-dimensional transform and then re-projected back to the original space. This is implemented for you in `PCA.reconstruction_error`.

k-Means clustering

k-Means clustering is a simple iterative method for finding clusters that alternates between assigning clusters to centers, and recomputing center locations given assigned points. Formally, we can describe the process as follows:

[2] This is easy to do using Numpy indexing. For a useful tutorial, see `https://docs.scipy.org/doc/numpy/reference/arrays.indexing.html`

[3] Note that unless $V_L = V$ (we use all of the columns of V), the reconstructed X'' won't necessarily exactly equal X. This difference is what `PCA.reconstruction_error` measures.

```
Data: number of clusters $K$, data $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$
Result: cluster centers, $\{\mathbf{c}^{(k)}\}_{k=1}^{K}$
Initialize each $\mathbf{c}^{(k)}$ to a randomly chosen $\mathbf{x}^{(i)}$
while not done do
    Assign each $\mathbf{x}^{(i)}$ to the closest $\mathbf{c}^{(k)}$
    Set each $\mathbf{c}^{(k)}$ to be the average of all the $\mathbf{x}^{(i)}$ assigned to it
    If cluster assignments did not change, done
end
```

Implement this algorithm in `KMeans.cluster`. Note that detecting whether cluster assignments have changed can be tricky if there are ties in the distances, so you may want to make use of the helper function `consistent_ordering`, or you may want to handle this in your own way.

Once you've implemented this method, the `KMeans` class should be complete. Typically, the cluster centers are what we're after in k-Means clustering, but these can be used in a variety of different ways. If the purpose is feature reduction/transformation, we can use distance to the discovered clusters as a new set of features. This is how the provided 2D and 3D plotting functions work: they first find two/three clusters, then transform the data into this new space, which can then be used as a visualization of the high dimensional data.

Alternatively, if visualization isn't the goal we can instead label each point with the closest cluster center, and then use the cluster centers themselves as a kind of "summary" of the data. It's difficult to come up with good measures of clustering performance when all we have is the data itself, but thankfully for each of the provided datasets we also have an actual class label (even though we ignored it when clustering). One way to measure cluster performance when we have access to this class label data is by measuring the Mutual Information between the cluster labels and the class labels. If the clusters do a good job of summarizing the data, then they should have a strong statistical relationship with the class labels. The standard statistical formula for measuring this kind of relationship is known as Mutual Information, and it's defined to be

$$MI(U,V) = \sum_i \sum_j p(U_i, V_j) \cdot \log \frac{p(U_i, V_j)}{p(U_i) \cdot p(V_j)}$$

where $p(U_i, V_j)$ is the joint probability of a datapoint having both cluster label i and class label j, while $p(U_i)$ and $p(V_j)$ are the respective marginal probabilities for cluster label i and class label j. The max and min values of Mutual Information are dependent on the number of class and cluster labelings, so this equation is often normalized to be a value between 0 and 1, with values near 0 having a weak relationship, and values near 1 having a strong relationship. This normalized version is implemented for you in `KMeans.normalized_mutual_information`. Try this out on the testing dataset with two clusters and you should see a Normalized Mutual Information very close to 1, indicating that the cluster labels agree with the class labels.

Questions

1. PCA

   (a) Run PCA on the three datasets, `iris`, `HTRU2`, and `digits`. For each dataset, visualize the 2D and 3D projections of the data (6 graphs). Make sure each point is colored according to its class label (see the "with labels" example in the template main function).

   (b) If we were to use PCA to transform the original features into those visualized in the previous question, do you think it would improve, reduce, or have no effect on classification performance? Explain your reasoning. Answer for both 2D and 3D for each of the three datasets.

2. k-Means

(a) Use k-Means as a feature transform, and visualize the transformed data in 2D and 3D for each of the three datasets (like you did for PCA). Again, use the "with labels" version so that the class label of each point is clear.

(b) Compare k-Means to PCA as a *feature transform* for each of the three datasets. Do you think it would be better to use PCA or k-Means to transform the data into 2D and 3D? Specifically, do you think it would be better in terms of classification performance on the transformed data? Explain your reasoning, and answer for both 2D and 3D for each dataset.

(c) The primary use for k-Means is not as a feature transform, but as a clustering method. Use the Normalized Mutual Information metric, described above, to get a quantitative measure of cluster quality when using the same number of clusters as classes for each of the three datasets. Report the performance, explain why you got the results you did.

3. Bayes Nets

(a) Consider the following four random variables: $Cough, Fever, Allergies, Flu$. You know that a cough can be caused by either allergies or the flu, fever can only be caused by the flu, and allergies and the flu have no causal relationship. Draw a Bayes Net that represents this set of random variables and their independence/dependence relationships.

(b) Using the Bayes Net from the previous question, write down how you would factor the joint probability distribution, $p(Cgh, Fev, Alg, Flu)$, as a product of conditional probabilities and marginals given by the Bayes Net.

(c) In standard Hidden Markov Models, the three distributions you need to do any of the inference tasks we talked about (filtering, smoothing, prediction, learning) are the *transition model*:

$$p(Z_t = k \mid Z_{t-1} = j) = A_{jk}$$

the *observation model*:

$$p(X_t = l \mid Z_t = j) = B_{jl}$$

and the *prior*:

$$p(Z_0 = j) = \pi_j$$

A simple modification of HMMs called Input/Output HMMs (or IOHMMs) changes the assumptions slightly. Instead of assuming transitions between hidden states are completely hidden, the model assumes that transitions can be at least partially influenced by some observable random variable, U_t. The Bayes Net for this model is given in Figure 2. Under the IOHMM model, what distributions do you need? Describe them precisely in probability notation as was done above for HMMs.

(d) In class we talked about how observing a common effect can make two marginally independent random variables not conditionally independent (conditioned on the common effect). A simple example of this is the `XOR` function on two bits. Let X_1 and X_2 be random variables that take the values 1 or 0 with equal probability, and let Y be a random variable which is the `XOR` of X_1 and X_2. Show that, given Y, the joint probability of X_1 and X_2 is not the same as the product of X_1 given Y and X_2 given Y. That is, show that

$$p(X_1, X_2 \mid Y) \neq p(X_1 \mid Y) \cdot p(X_2 \mid Y)$$

For reference, the Bayes Net for this problem is given in Figure 3.

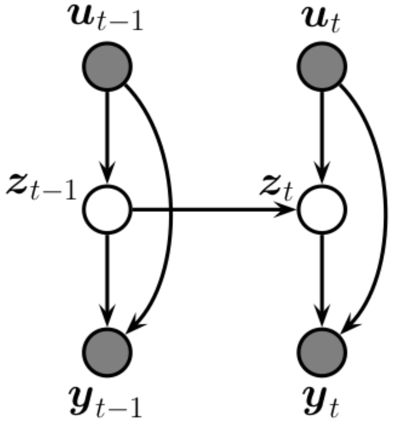
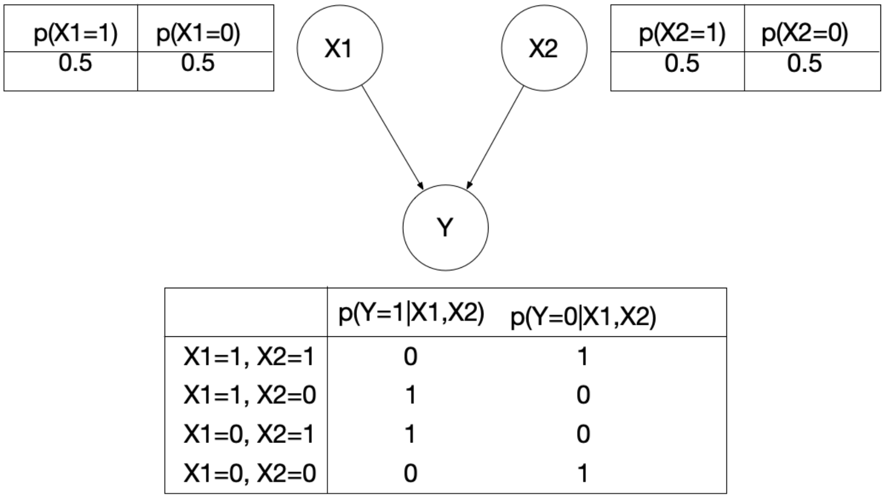


Figure 2: Bayes Net for IOHMMs.



p(X1=1)	p(X1=0)
0.5	0.5

p(X2=1)	p(X2=0)
0.5	0.5

	p(Y=1|X1,X2)	p(Y=0|X1,X2)
X1=1, X2=1	0	1
X1=1, X2=0	1	0
X1=0, X2=1	1	0
X1=0, X2=0	0	1

Figure 3: Bayes Net for the `XOR` question.