Assignment

In this assignment, I experiment several unsupervised learning techniques on two datasets:

1) Clustering techniques: K-Means and Expectation-Maximation
2) Dimensionality Reduction: Principal Component Analysis, Independent Component Analysis, Randomized Projects, and Linear Discriminant Analysis

Lastly, I experiment using the features generated from the above unsupervised learning techniques in a supervised learning setting such as Neural Network as an additional feature.

Datasets Descriptions

Two datasets are used for this assignment:

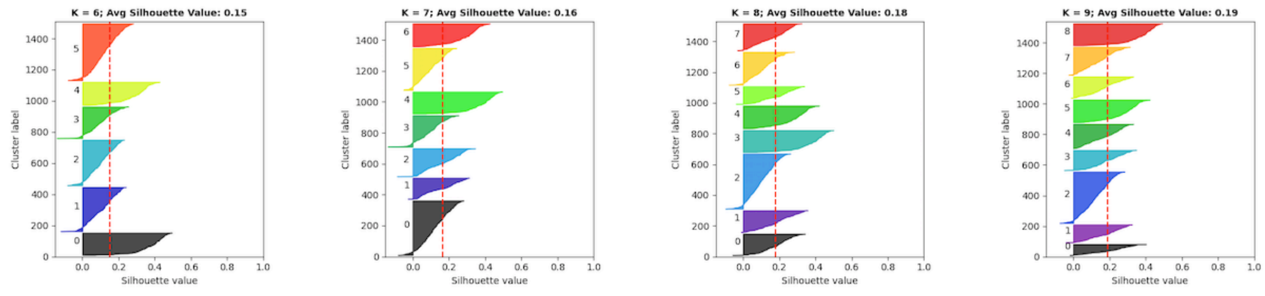
1) **MNIST dataset** provided by sikit-learn. Each input is represented by 64 pixels in 2-dimensional (8,8) format. Each pixel is represented by an integer that's in the range of 0 to 16 (0 is complete white, 16 is complete dark). Each label is represented by an integer in the range of 0 to 9, indicating the digit that the input 64 pixels represents. There are total of ~1800 input/label datapoints, of which 80% are used for training, 20% are used for testing.
2) **Wine dataset** provided by sikit-learn. Each input is represented by 13 scalar wine features including color intensity, amount of magnesium, etc. Each label represents the type of wine. There are total of 3 unique types of wine in the dataset. There are total of ~180 input/label datapoints, of which 80% are used for training, 20% are used for testing.

Clustering: K-Means

An important problem to solve for K-Means is to determine which k to use for a dataset, without knowing the labels. A useful technique to assess a chosen k is to study the separation between the resulting clusters, with the preference to choose k that results in higher inter-cluster distance and lower intra-distance. Silhouette score is used to measure how close a particular data point is to the other data points within its assigned cluster and how far a particular data point is to the other data points in the nearest neighboring cluster. A +1 silhouette score means that a particular data point is far from its neighboring cluster and close to the data points within its own cluster. A -1 silhouette score means that the data point is close to its neighboring cluster and far to the data points within its own cluster, which likely means that this data point has been misclassified. For both MNIST and Wine dataset, I iterate through different number of k, and calculate the average silhouette scores for all the datapoints in the training set.

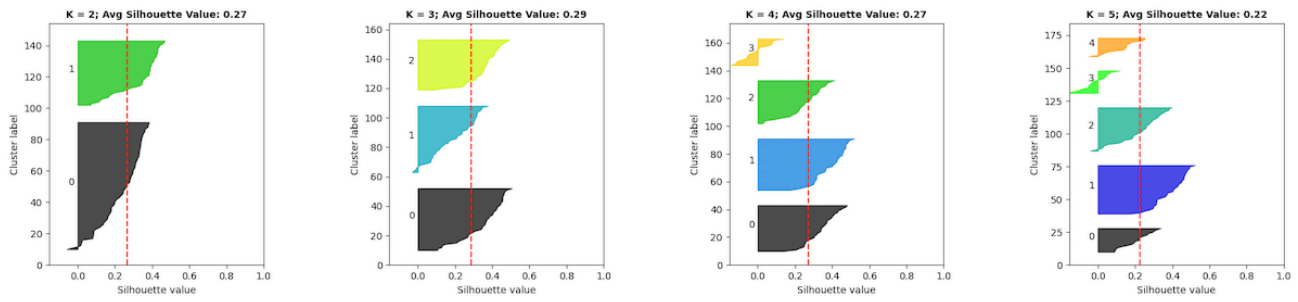
MNIST: Silhouette values for various k

Only showing k from 6 to 9 to save space. Real study done k from 2 to 20



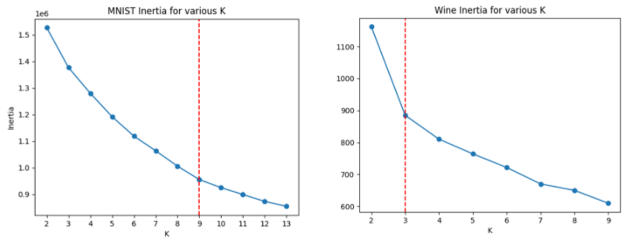
Wine: Silhouette values for various k

Only showing k from 2 to 5. Real study done k from 2 to 20



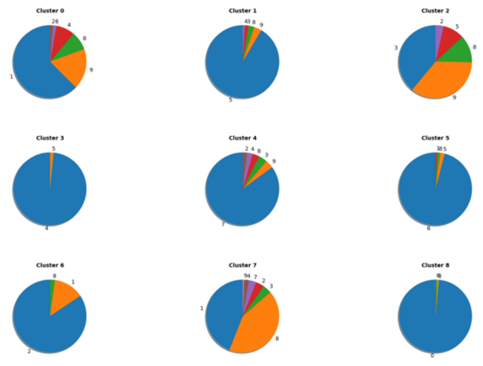
In choosing k, I consider two factors: 1) high average silhouette value, and 2) each cluster has a well-balanced silhouette score, for example, there's no particular cluster where the majority data points have low silhouette score. Therefore, based on silhouette charts, k of 9 seems promising for MNIST and k of 3 seems promising for Wine.

Inertia, sum of squared distances of each datapoint to its respective centroid, is another metric to assess k, with the intuition data points should be as close to their respective cluster's centroid as possible. Below I plot the value of inertia for different k, for both MNIST and WINE datasets.



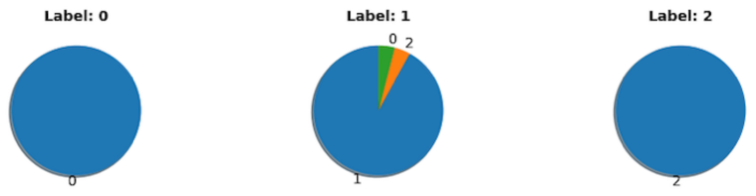
Looking at the inertia charts left and using the elbow (point of inflection at which increasing k does not give much better result) method, inertia chart also suggests that k of 9 should be chosen for MNIST and k of 3 should be chosen for WINE.

Once we have chosen k, and since we have the actual labels (reminder that the above clustering is done without utilizing the labels), let's see how the resulting clusters line up with the actual labels. There are two metrics that can be used for this assessment: 1) *homogeneity*, which measures how "homogenous" each resulting cluster is, based on the labels; if each cluster's data points are all of the same label, then that is the best case, 2) *completeness*, which is the opposite of homogeneity; for all the data points with the same label, if they are assigned the same cluster, then that's the best case. To demonstrate, I will provide a visualization of *homogeneity* for MNIST and a visualization of *completeness* for Wine:



Left chart represents *homogeneity* for MNIST using k of 9 clusters. As shown, within each cluster, by using the true labels, we can see how "homogenous" each cluster is. Cluster 3, for example, achieves the most "homogeneity" because all except few data points are digit 6. Cluster 2, for example, achieves the least "homogeneity" because it contains a lot of digits with significant portion within that cluster.

I will provide a table that summarize "homogeneity" and "completeness" score at the end of this assignment to compare among different clustering technique.



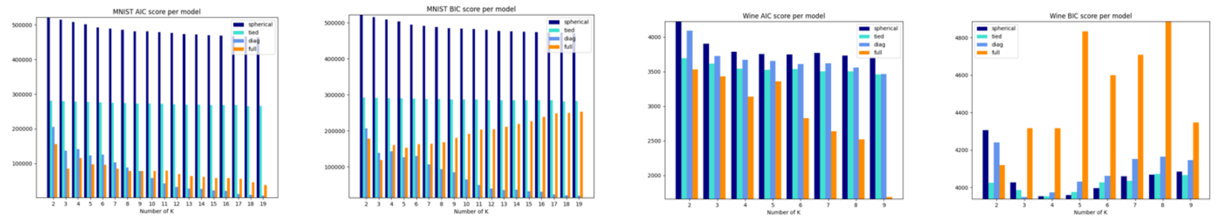
Left chart represents *completeness* for Wine using k of 3. As shown, all the data points with label of 0, are assigned to the same cluster 0. That means data points with class label 0 achieves high "completeness". On the other hand, data points with class label 1 are scattered among cluster 0, 1, and 2, resulting a relatively lower "completeness".

Clustering: Expectation-Maximization (Gaussian Mixture)

K-means is a clustering algo that does not allow cluster to overlap. On the other hand, Expectation-Maximization allows assigns probability distribution to each data point in cluster assignment, also known as "soft-clustering". Each of the k cluster is a Gaussian model with a mean and a variance. The goal of Expectation-Maximization is to find a set of clusters, such that, the likelihood of the observed data points produced by this set of clusters is the highest. There are two metrics that I calculate in picking the right k for Expectation-Maximization: Akaike Information Criterion (AIC), and Bayesian Information Criterion. They both describe how well a given model fits a set of observed data and how complex the model is. The goal is to *minimize* AIC and BIC as they are the sum of (-1) * model_fit + model_complexity_penalty. They resembles Minimum Description Length in a sense that we want a model that fits the data well, yet not too complicated (Occam Razor). The difference the degree that AIC and BIC penalizes a model's complexity. AIC's penalty of a model's complexity is 2*k where k is the model degrees of freedom, where as BIC's penalty of a model's complexity is ln(N)*k where N is the number of observed data points. As shown, as N, number of observed data point increases, ln(N) will increase and overcome 2, which is a static multiplication term in AIC's penalty for model complexity. Therefore, using AIC is more likely to cause overfitting, whereas using BIC is more likely to cause underfitting. In practice, both metrics are looked at in determining a model's parameter.

Another important factor to consider is choosing the covariance type in the model: full, or tied, or diagonal, or spherical. Below, I calculated AIC and BIC for different number for k and covariance type, for both Wine and MNIST dataset.

Left Two Charts: MNIST, AIC and BIC for various k and covariance type
Right Two Charts: Wine, AIC and BIC for various k and covariance type



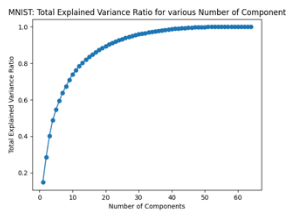
For MNIST charts above, it seems that spherical and tied covariance type all produce really high AIC and BIC. It makes sense the both "spherical" and "tied" covariance types extremely limit the "expressiveness" of each gaussian mixture, for example, "tied" specified that each gaussian mixture should have the same shape, and "spherical" specifies that each gaussian mixture should be of circular contour. Therefore, I focus on "diagonal" and "full" covariance type. "diagonal" allows each gaussian mixture to have any shape as long as their axes are oriented along the coordinates, whereas “full” allows each gaussian to not only have any shape but also any direction (axes do not have to be oriented along the coordinates). They produce similar AIC score. However, BIC score for “full” covariance increases significantly as we increase k, it makes sense because as mentioned, BIC penalizes a model’s complexity more heavily, and “full” covariance requires significantly more parameters. Therefore, I go with “diagonal” covariance type and pick k of 12, after which, the additional increase in k does not reduce AIC or BIC that much. Following similar thought pattern and looking at Wine’s AIC and BIC chart, k of 3 seems to be a good pick as after which, AIC or BIC’s score does not decrease.

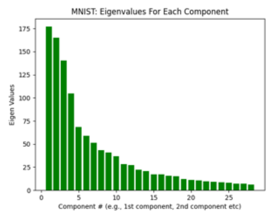
As mentioned, the homogeneity and completeness scores that measure how clustering “lines up” with true labels will be summarized in a table for comparison at the end.

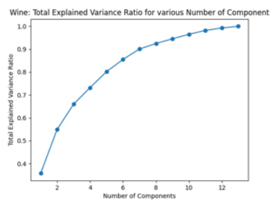
Dimensionality Reduction: Principal Component Analysis

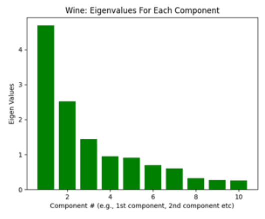
Principal component analysis finds the first component (a direction in the input space), such that the projections of datapoints onto that component contains the maximum variance (the amount of variance is represented as eigenvalues). Then, it finds the second component, which *has to* be orthogonal to the first component, that maximizes the variance of datapoints’ projection. The intuition is that there’s correlation among the features in the original space, by rotating the features into a new space, we can make the projections in the new space statistically uncorrelated. This rotation is likely to allow the reduction of dimensions needed to represent the data in a way loses negligible information, because redundant information (e.g., correlated features) is decreased in the new space and less parameters are needed to model the data. *Explain ratio* is defined as the sum of the variances (eigenvalues) represented by all the components in the new space divided by the total variance. Intuitively, it represents how much variance does your new space (of k components) captures. In choosing a k for both MNIST and Wine datasets, I pick a threshold of 95% explain ratio and pick the lowest k that reaches the 95% ratio.

Left Two Charts: MNIST, Explained Variance Ratio for various k
Right Two Charts: Wine, Explained Variance Ratio for various k



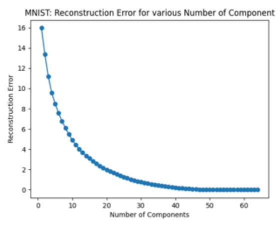


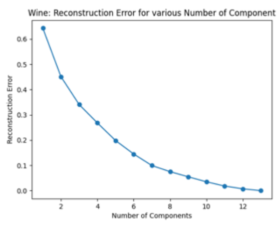




For MNIST dataset, there are 64 dimensions in the original space as there are 64 pixels. Therefore, in the exercise, I varied the k from 1 64. You can see a very steep climbing curve initially, it makes sense because as mentioned, the first component should have the largest variance (eigenvalue), then the second component should have the second largest variance, etc. The curve plateaus after k of 30 because very little variance is left to be captured in additional dimensions which can be discarded. Therefore, almost 30 dimensions (half of the original 64 dimensions) can be discarded. On the other right, for Wine, we need k of 10 components (the original space has 13 dimensions / features, therefore, most of the original dimensions) to meet the 95% threshold. It makes sense that because, for MNIST dataset, there are a lot of correlation in the image, for example, if there’s a pixel that is black somewhere in the middle of the (8,8) image, then it’s likely that particular pixel’s surrounding pixels are also dark. Whereas in wine dataset, there doesn’t seem to be that much correlation among its original 13 features.

It's also worth to point out, that, as you change the number of k, the value first, second.. principal components' eigenvalue does not change, as it always looks to maximize the variance in finding each principle component.



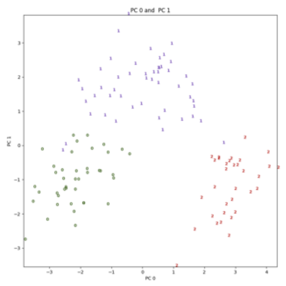


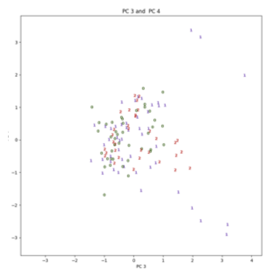
Another metric to assess k is to check "how much" information did the transformation / dimensionality reduction lose by reconstruct the transformed point back into the original space and compare the distance between the reconstruction and the original datapoint's location. See left charts the reconstruction error chart for various k for both MNIST and Wine. It's no surprise that as the number of k increases, the "reconstruction error" or "information loss" decreases. For MNIST particularly, after k of 30, any increment of k does not result in much loss of information.

For a more visually intuitive dataset such as MNIST, we can, for each digit from 0 to 9, pick and visually compare the before transformation vs. reconstructed after transformation (almost identical visually!):

Original:

Reconstruction after transformation:

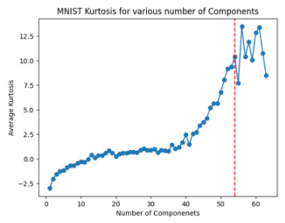


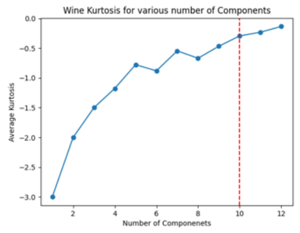


We can also visualize the degree of variance decreases as the principle component increases. Take Wine dataset for example, let's plot what the transformed dataset looks like in principle component 0 and 1 vs. principle component 3 and 4. See chart on the left. We can see that data point's values principal component 0 and 1 are a lot more "dispersed" with more variance than their values along component 3 and 4.

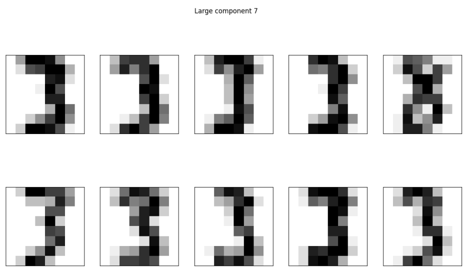
Dimensionality Reduction: Independent Component Analysis

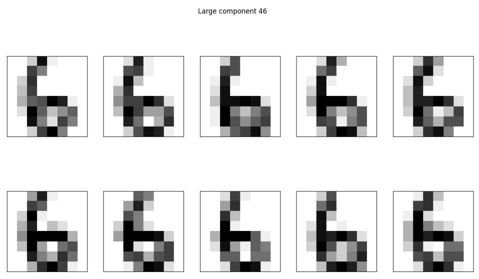
When the observed datapoints are the linear combination of some hidden signals, such as cocktail party problem, Independent Component Analysis can be used to find uncover these hidden signals. A necessary condition for Independent Component Analysis to work is that the hidden signals are non-Gaussian and statistically independent. If the hidden signals are gaussian, we might as well use Principle Component Analysis instead. The key here is to figure out how to measure a signal's non-Gaussian-ness. A common metric that is used is Kurtosis, which measures the "tailedness". In Fisher's kurtosis definition, kurtosis of 0 represents normal distribution, -3 represents uniform distribution, and kurtosis greater than 0 means "fatter" tails and higher peak. Therefore, we would like to choose a k that produces high average kurtosis for each dimension of the transformed data.





Left charts illustrate the average kurtosis as number of k increases for Independent Component Analysis, for both MNIST and Wine datasets. As shown, when component equals 1, the kurtosis is -3 because if you only use 1 component then the distribution along that component after transformation has to be uniform. MNIST's average kurtosis reaches ~10 as k increases whereas Wine's average kurtosis doesn't even reach 0 as k increases. It makes intuitive sense that MNIST images can be thought of made up by image features, for example, edge, corner etc. For MNIST's each independent component, I find the set of original images that achieves the highest project on that component:



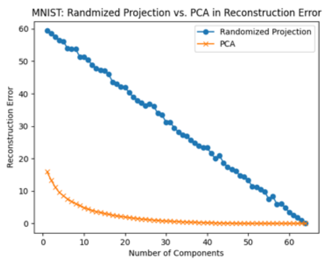


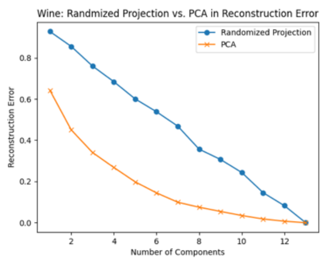
As shown above, the set of original images that achieves the highest value in independent component #7 is a set of 3s, and the set of original images that achieves the highest value in independent component #46 is a set of 6s. That means, these components that we found can be used as feature detector in digit image classification algorithms.

It's worth noting that, to the contrary of Principle Component Analysis that Independent Component Analysis' first, second… component does change, as k varies for MNIST and Wine.

Dimensionality Reduction: Random Projection

When the dimension space gets higher and higher, there are more and more "*almost*" orthogonal projections such that, by chance, a randomly generated vector can be "close" to an orthogonal directions. Johnson-Lindenstrauss lemma stats that, given the data set *n*, and a desired error threshold *e*, the minimum number of dimensions that we can project to with high probability based on random projection is *p > ln(n) / e^2.* Assuming an error threshold of *e* = 0.2, since the MNIST dataset has ~1440 data points for training, then the minimum number of dimensions needed for MNIST is *ln(1440) / 0.2^2* = 180 dimensions and since the Wine dataset has ~144 data points for training, then the minimum number of dimensions needed for Wine is *ln(144) / 0.2^2* = 125 dimensions. However, the MNIST dataset contains only 64 dimensions and Wine dataset contains only 13 dimensions. Therefore, the practical use of random projection probably makes more sense when there are a lot of dimensions to work with. The calculation of random projection is extremely fast because we do not need to calculate covariance or eigenvalue decomposition as needed in PCA.



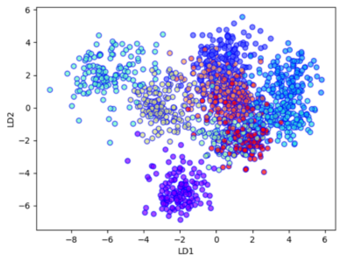


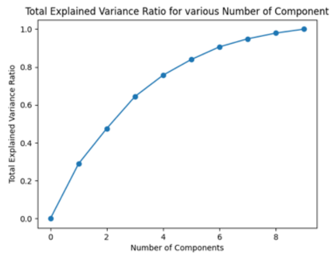
Left chart compares Randomized Projection's reconstruction error vs Principle Component Analysis' reconstruction error as number of component k varies. For each run of Randomized Projection of k, I average the result over 30 runs to and calculate the average of reconstruction error over these 30 runs, for each k. It's no surprise that the reconstruction error decreases in a straight / linear line because, because each with increment of k, the algorithm adds a random direction, whereas Principle Component "purposely" add a direction that maximizes variance (hence the drastic decrease in the beginning of the principle component curve). I also investigates the variation of 30 runs result as k increases, it's interesting that the variation first decreases as k increases, and then decreases as k approaches the number of dimensions in the original space.

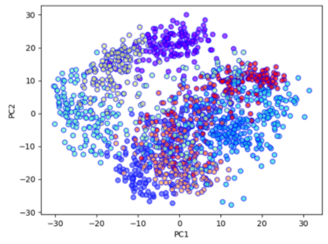
Dimensionality Reduction: Linear Discriminant Analysis

In contrast to all the dimensionality reduction algorithms experimented above, Linear Discriminant Analysis actually utilizes the class labels, to transform the datapoints in a new space such that it maximizes the separation between classes. The dimension of the output *has to be* less than number of classes.

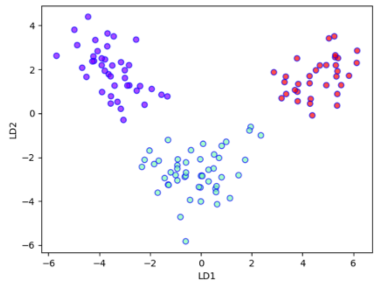
Left: MNIST, LDA transformed data projected to LD1 and LD2
Middle: MNIST LDA Total Explained Variance Ratio for various k
Right: MNIST, PCA transformed data projected to PC1 and PC2

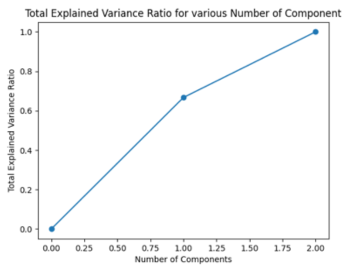


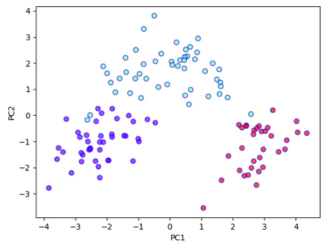




Left: Wine, LDA transformed data projected to LD1 and LD2
Middle: Wine LDA Total Explained Variance Ratio for various k
Right: Wine, PCA transformed data projected to PC1 and PC2







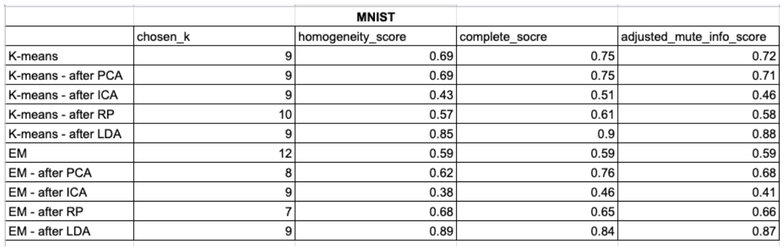
As shown above for both MNIST and Wine datasets, the datapoints in Linear Discriminant Analysis projected space form very nice clusters that are far away from each other, farther away than that of the datapoints in Principle Component Analysis projected space, because Principle Component Analysis focuses increasing the variance instead.

Similar to Principle Component Analysis, Linear Discriminant Analysis' first, second… component does change, as k varies for MNIST and Wine.

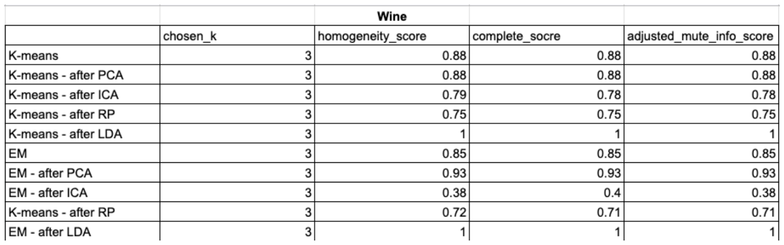
In addition, MNIST Linear Discriminant Analysis achieved 95% accuracy in the 20% held-out test set, and Wine Linear Discriminant Analysis achieved 98% accuracy in the 20% held-out test set.

Apply Clustering after Dimensionality Reduction

After applying the above four dimensionality reduction algorithms on MNIST and Wine, we now apply K-Means and Expectation-Maximization clustering techniques on the output of these dimensionality reduction transform. Similar to how we choose k for a clustering algorithm above, k is also individually tuned for every clustering algorithm (for K-Means, investigate silhouette score and for Expectation-Maximization, investigate AIC and BIC score) applied on the output of every dimensionality reduction algorithm. Also as mentioned, we use homogeneity and completeness score to assess how well the resulting cluster "line-up". Below charts illustrates the performance:



		MNIST		
	chosen_k	homogeneity_score	complete_socre	adjusted_mute_info_score
K-means	9	0.69	0.75	0.72
K-means - after PCA	9	0.69	0.75	0.71
K-means - after ICA	9	0.43	0.51	0.46
K-means - after RP	10	0.57	0.61	0.58
K-means - after LDA	9	0.85	0.9	0.88
EM	12	0.59	0.59	0.59
EM - after PCA	8	0.62	0.76	0.68
EM - after ICA	9	0.38	0.46	0.41
EM - after RP	7	0.68	0.65	0.66
EM - after LDA	9	0.89	0.84	0.87



		Wine		
	chosen_k	homogeneity_score	complete_socre	adjusted_mute_info_score
K-means	3	0.88	0.88	0.88
K-means - after PCA	3	0.88	0.88	0.88
K-means - after ICA	3	0.79	0.78	0.78
K-means - after RP	3	0.75	0.75	0.75
K-means - after LDA	3	1	1	1
EM	3	0.85	0.85	0.85
EM - after PCA	3	0.93	0.93	0.93
EM - after ICA	3	0.38	0.4	0.38
K-means - after RP	3	0.72	0.71	0.71
EM - after LDA	3	1	1	1

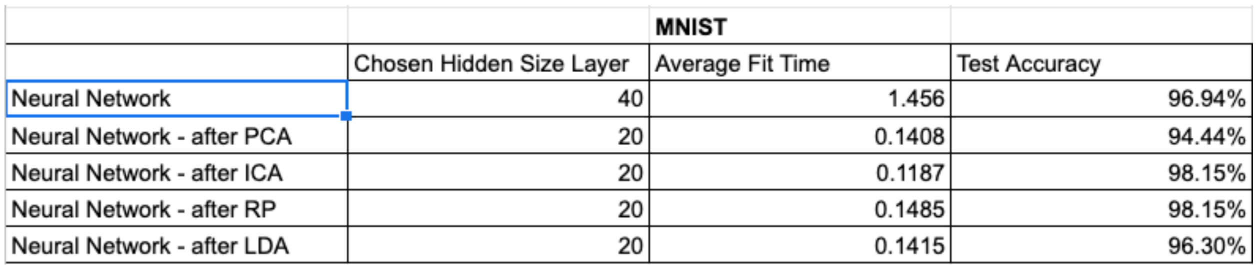
A few observations for MNIST and Wine results above: 1) Clustering algorithm applied after Principal Component Analysis seems to do better (definitely not worse) than clustering algorithm without dimensionality reduction, 2) Clustering algorithm applied after Independent Component Analysis seems to do worse than clustering algorithm without dimensionality reduction, 3) While Clustering algorithm applied after Random Projection's performance is very "close" to that of clustering algorithm without dimensionality reduction. 4) It should be of no surprise that clustering algorithm applied after Linear Discriminant Analysis should perform better than clustering algorithm without dimensionality reduction, because Linear Discriminant Analysis utilizes the true labels to purposely transform the data points so the different labels are as far away from each other as possible.

Apply Neural Network after Dimensionality Reduction

After applying the four dimensionality reduction algorithms on MNIST, now we apply a neural network architecture given the transformed / dimensionality reduced dataset. In assignment 1, we applied neural network architecture of 1 hidden layer of size 40 on the MNIST dataset and achieved 96.96% accuracy on the 20% held-out test set, which we use as a baseline performance for comparison below.

Since the dataset has been transformed in a new dimension with lower number of dimensions, we first need to tune the neural network's parameters, using cross validation. We find that the hidden size layer required for dimensionality reduced dataset reduced from the original 40 to 20. It makes sense because once the dimensionality reduces, it's likely that the model complexity required to train the testing data well decreases as well.

Below table summarizes the training speed and testing accuracy:



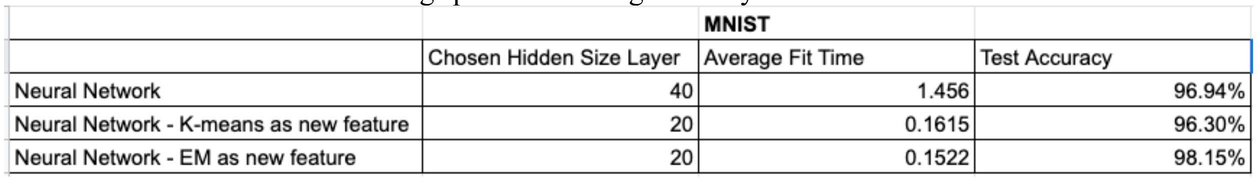
		MNIST	
	Chosen Hidden Size Layer	Average Fit Time	Test Accuracy
Neural Network	40	1.456	96.94%
Neural Network - after PCA	20	0.1408	94.44%
Neural Network - after ICA	20	0.1187	98.15%
Neural Network - after RP	20	0.1485	98.15%
Neural Network - after LDA	20	0.1415	96.30%

As shown, the average training / fit time required for transformed data significantly reduced because of a simpler model and lower dimension in the training data. The test accuracy for are also in-line with, if not better, than the original neural network trained on un-transformed data.

Apply Neural Network after Clustering

Now we use the generated label from K-Means and Expectation Maximization clustering algorithm as a new feature in addition to MNIST original 64 features. Then apply neural network algorithm. Similar to above, we use a similar neural network architecture of a 1 hidden size of 40 as a baseline. Although the dimensionality does not decrease this time (the dimensionality even goes up by 1), we still see that ~20, instead of 40, hidden size is enough during parameter tuning (using cross validation). This could be explained that the "clustering information" as an attribute provides a lot of "information gain" and the resulting model complexity can be simpler because it. Similar to how in a decision tree structure, if you have an attribute that provides a lot of "information gain", the tree depth can be shorter because of it.

Below table summarizes the training speed and testing accuracy:



		MNIST	
	Chosen Hidden Size Layer	Average Fit Time	Test Accuracy
Neural Network	40	1.456	96.94%
Neural Network - K-means as new feature	20	0.1615	96.30%
Neural Network - EM as new feature	20	0.1522	98.15%

As shown above, introducing K-means or EM as new feature both reduce average fit time while maintain high test accuracy as the baseline.