Harsh Patel

CS 4641 – Machine Learning

Assignment 3: Unsupervised Learning

Datasets Background:

PenDigits Dataset:

The *PenDigits* dataset was created by collecting 250 samples from 44 different writers. Each writer wrote the digits zero through nine (in random order for random multiple of times for each digit until 250 samples were collected) and those images were scanned and encoded to convert image data into multivariate data. The overall goal of the algorithms will be to classify these encodings into one of the ten digits (zero through nine). The dataset contains 10992 instances with 16 attributes for each instance.

Vehicle Dataset:

The *Vehicle* dataset was created by taking 360° images of different types of vehicles and then extracting the silhouettes of the vehicles from each image. In other words, this dataset focuses on 2D shape and edge detection extracted from 3D objects. There are 846 instances with 18 attribute per instance. Examples of attributes include: compactness, circularity, radius ratio, and skewness about major and minor axes. The overall goal of the algorithms will be to classify each instance into one of four classes ("opel", "saab", "van", and "bus").

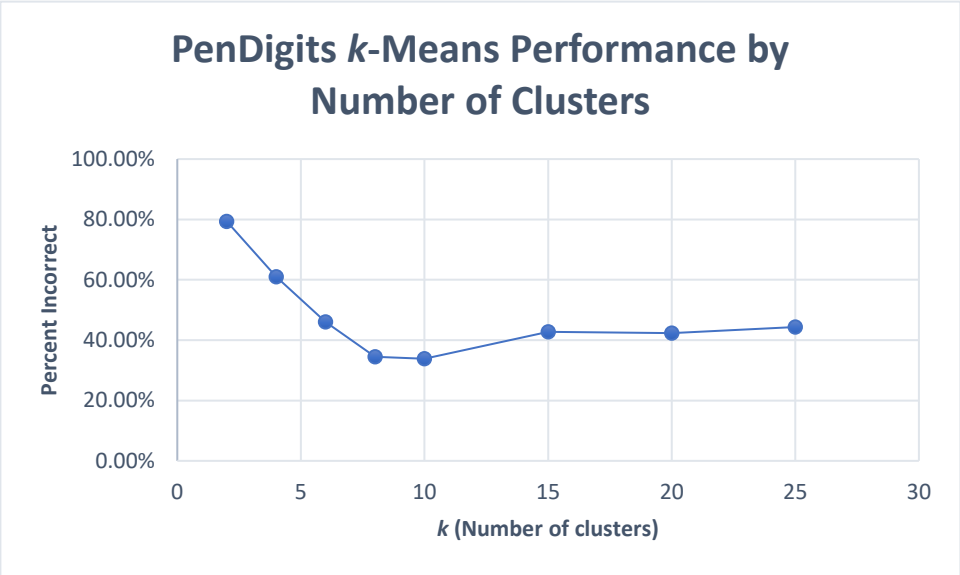
[Note: Refer to Assignment 1 for a more detailed background on the datasets]

Clustering Algorithms:

k-Means:

The *k*-Means algorithm creates hard clusters by randomly initializing *k* cluster centroids and then assigning each instance in the dataset to one of these *k* clusters using a selected distance metric (i.e. Euclidian Distance). After all instances are assigned to exactly one cluster, the *k* cluster centroids are then updated to reflect the average of the instances within that cluster. This is performed over and over until convergence. To explore the *k*-Means algorithm with respect to the PenDigits and Vehicle datasets, I utilized the "SimpleKMeans" algorithm (using Euclidian distance as the distance metric) built into WEKA (Waikato Environment for Knowledge Analysis, a software package) and observed how classification error changed as I varied the number of clusters (*k*).

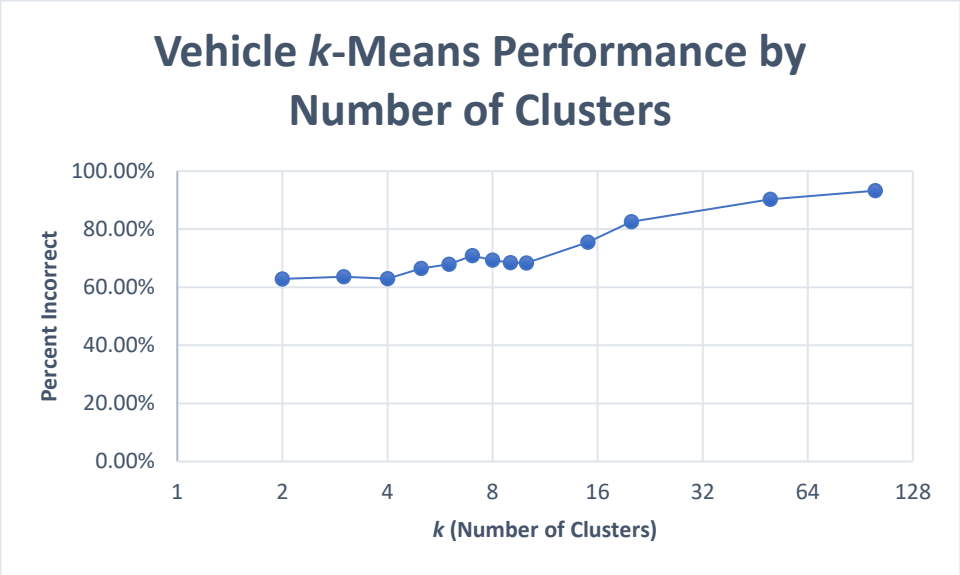
For the PenDigits dataset, I used the following values for *k*: 2, 4, 6, 8, 10, 15, 20, and 25, recording within cluster sum of squares (cohesion of cluster), number of iterations till convergence, and percent of instances incorrectly clustered (using WEKA's Classes to Clusters Evaluation Metric). The results for that experiment are below:



Number of Clusters	Within cluster sum of squared errors	Iterations
2	12849.99546	38
4	8525.470651	22
6	6988.630442	13
8	5639.51642	13
10	5079.917341	22
15	4065.84833	93
20	3470.918522	56
25	3137.153624	43

From the figure above, percent error (percent inaccuracy) initially decreases as the number of clusters increases and this makes sense as you are allowing more splits to occur within your dataset creating more specific clusters. When $k = 10$, we observe the lowest percent error value, interestingly corresponding to the number of different labels in the PenDigits dataset (labels include the integers between 0 and 9). We also see a comparable percent error value around $k = 8$, indicating certain instances with different labels have similar Euclidian distances with respect to each other and with respect to the generated cluster centroids. From domain knowledge of the dataset, we know several labels have inherent similarity bias, such as the handwritten integers three and nine as well as the handwritten integers zero and eight, which could explain why having lower clusters can still capture a large percent of the underlying differences between the instances. Moving on, we see an increase in percent error as we increase the value of k past 10 and eventually tapering off as k is around 25. Overfitting could be the cause of such a dilemma where specific generated cluster centroids are moving all the time to perfectly find a convergence location (backed by the table above which shows larger number of iterations as k reaches high values), but as a result, the final clusters are mostly similar but still have distinct irregularities resulting in a higher error percentage. Finally, looking at the within cluster sum of squares (cohesion within clusters), we see a decline in that cohesion value as the number of clusters increases and this is due to the increasing homogeneity of the clusters as more clusters centroids are generated allowing for better separation of instances (ultimately leading to few differences in instances within each cluster).

For the Vehicle dataset, I used the following values for k: 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 50, and 100, recording the same metrics as above during the PenDigits experiment. As the Vehicle dataset has fewer instances than the PenDigits dataset, I was able to allow higher values of k and not have to wait a long time for the algorithm to run (a problem I faced when trying to run very high values of k for the PenDigits dataset). The results for the k-Means Vehicle experiment are below:



Number of Clusters	Within cluster sum of squared errors	Iterations
2	315.5856735	9
3	250.9362497	7
4	223.5402089	8
5	204.8052303	22
6	190.0413781	31
7	178.7787335	35
8	167.881328	23
9	158.7201603	16
10	153.6420063	24
15	120.8481737	19
20	103.4969026	18
50	69.8958307	13
100	50.0777303	13

Unlike the PenDigits dataset, the Vehicle dataset did not show an initial decline in percent error, but rather had comparable percent error values up until $k = 4$ (the number of labels in the dataset ["opel", "saab", "bus", and "van"]). After $k = 4$, the percent incorrect values continued to generally increase all the way till $k = 100$. One explanation for the comparable error values between $k = 2$ and $k = 4$ could be the similarity between the "opel" and "saab" silhouettes, as noted by the authors of the dataset. Having less than or equal to four clusters could have just grouped instances from those labels together resulting in similar error values regardless of the number of clusters. The number of iterations and within cluster sum of squares between $k = 2$ and $k = 4$ did decrease, as evident in the table above, indicating more homogenous clusters and a marginally better performance time at $k = 4$. The really interesting trend occurs after $k = 4$ and all the way till $k = 100$, where the percent error continues to increase (similar to the PenDigits dataset), but the number of iterations increase sharply and then gradually decrease. An explanation for this phenomenon could include the fact that after a certain number of clusters, the physical arrangement of the instances in space could be captured in a shorter amount of time due to the original homogeneity of labeled clusters. If the instances were already located close to each other and were able to be separated easily, then it would not take longer for more clusters to reach their convergence locations.

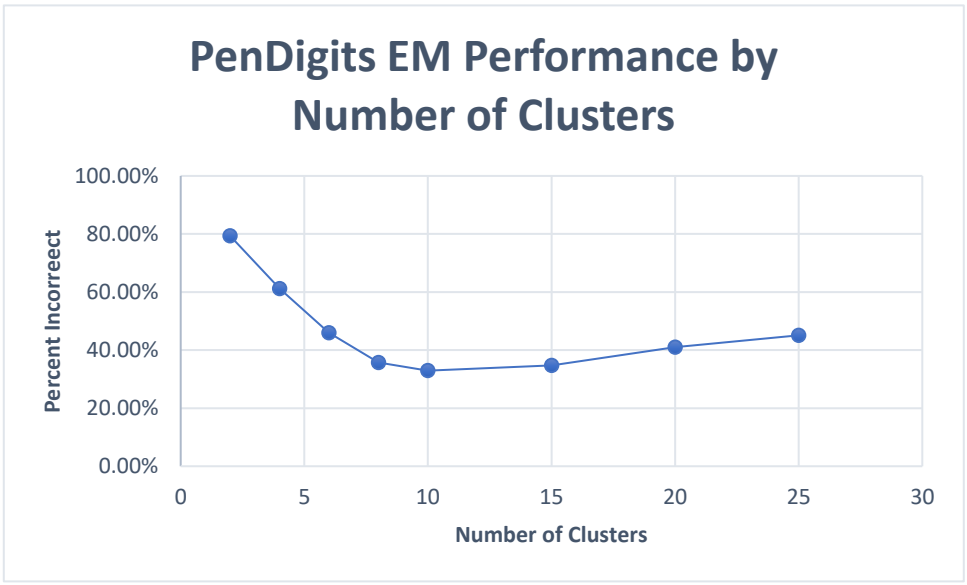
Overall, both datasets performed reasonably well on *k*-Means and their performance could be improved by adding more examples for the Vehicle dataset and adding better distinct examples for confusing handwritten integer pairs for the PenDigits dataset.

Expectation Maximization:

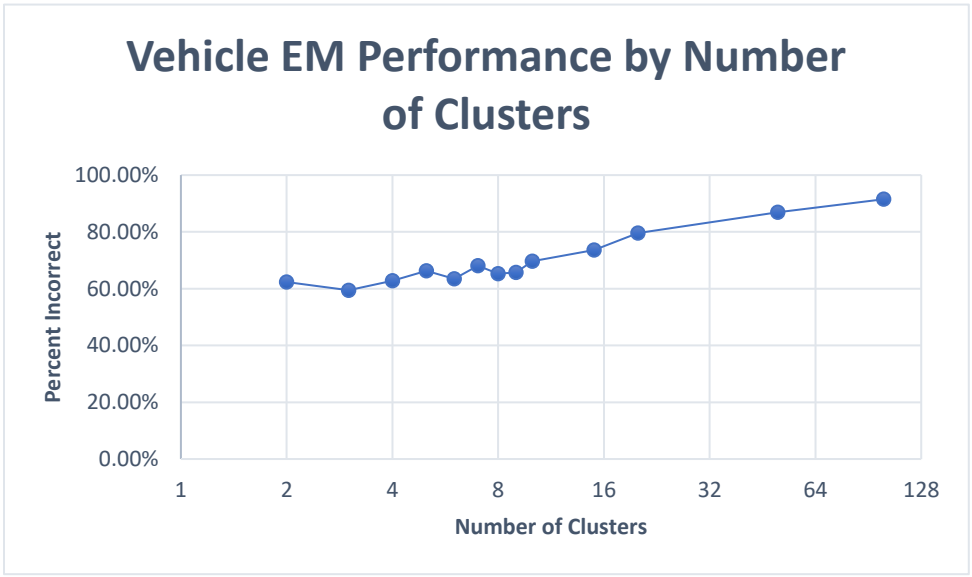
The Expectation Maximization (EM) algorithm creates soft clusters by randomly initializing normal distributions with means and variances that aim to allow one instance to be a part of more than one cluster based on the probability that instance value falls within the cluster normal distribution. To explore the Expectation Maximization algorithm with respect to the PenDigits and Vehicle datasets, I utilized the "EM" algorithm built into WEKA and observed how classification error changed as I varied the number of clusters.

For the PenDigits dataset, I used the following values for the number of clusters: 2, 4, 6, 8, 10, 15, 20, and 25, recording log likelihood, number of iterations till convergence, and percent

of instances incorrectly clustered (using WEKA's Classes to Clusters Evaluation Metric). For the Vehicle dataset, I, instead, used the following values for the number of clusters: 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 50, and 100, recording the same metrics as with the PenDigits dataset. Similar to *k*-Means, I used larger values for the number of clusters as the Vehicle dataset is relatively small and was able to handle the algorithm generating large number of clusters. The results for both experiments are shown below:



Number of Clusters	Log likelihood	Iterations
2	-74.30223	20
4	-70.2079	7
6	-68.01182	2
8	-67.55839	2
10	-66.95693	1
15	-64.21985	1
20	-62.25359	2
25	-60.35404	1



Number of Clusters	Log likelihood	Iterations
2	-64.06785	6
3	-62.20207	35
4	-60.85744	20
5	-59.4428	92
6	-57.94195	14
7	-57.80027	23
8	-56.1425	1
9	-55.69992	50
10	-55.0743	1
15	-53.48275	1
20	-51.66442	9
50	-47.97334	6
100	-46.1558	4

Looking at the PenDigits EM results, we see a similar trend for percent error as we saw with the *k*-Means algorithm. The minimum for percent error is around 10 clusters (equal to the number of labels in the PenDigits dataset) and the error continues to increase after 10 clusters. The log-likelihood value, measure of homogeneity within clusters, appears to increase as the number of clusters increase, which is what we could expect as the clusters become more and more refined. One interesting difference from the *k*-Means algorithm is that the number of

iterations decreases as the number of clusters increases. As EM allows for soft clustering, it has a much more permissive nature by allowing instances to belong to more than just one cluster. As a result, as the number of clusters increases, the overall algorithm has to run for less iterations as specific instances are not forced to jump between one cluster Normal Distribution to the next as they are allowed to belong to multiple distributions. Furthermore, this eases the pressure on the normal distribution as it does not have to change its mean and variance often since there are enough other distributions to capture all of the instances.

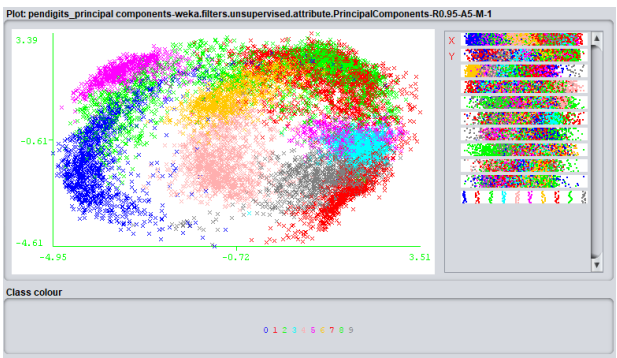
Moving on the Vehicle EM results, we actually observe very similar results as the *k*-Means algorithm, where low cluster numbers (between two and four) have comparable percent error values and cluster numbers greater than four have increasing percent error. Once again, this may be due to the lack of enough examples in the dataset or locality bias of instances discussed earlier in the *k*-Means section of this analysis. The log likelihood values continue to decrease as the number of clusters increase, similar to the PenDigits dataset. Unlike the *k*-Means experiment, the EM Vehicle experiment reports lower iteration values as the number of clusters approach 100. This is probably once again due to the soft clustering nature of the EM algorithm discussed in the paragraph above.

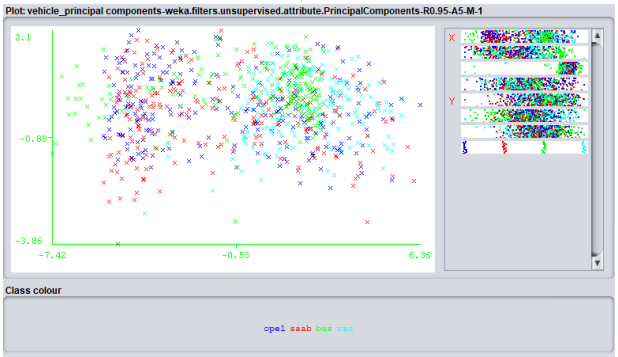
Overall, the differences between the two clustering algorithms are really apparent when looking at the number of iterations required and the values of homogeneity (within cluster sum of squares vs. log likelihood). For EM, I believe performance could be improved if the Vehicle dataset had more instances and if we selected a less stringent amount for minimum improvement in log likelihood for the PenDigits dataset as that would increase the number of iterations leading to better refinement of clusters.

Dimensionality Reduction Algorithms:

Algorithms like Principal Component Analysis, AutoEncoding, and Random Projections belong to a class of algorithms called Dimensionality Reduction Algorithms. These algorithms aim to reduce the number of features/number of attributes each instance has while trying to balance information loss and maximize total variance explained. Using WEKA's built in unsupervised attribute filter function, I applied PC (Principal Components) at different levels of variance explained, MLPAutoencorder (AutoEncoding) with different number of hidden units, and RandomProjections with varying number of dimensions my original data would be reduced to, performance evaluations of these changes are discussed further in the next component of this analysis when we apply clustering algorithms to the data transformed by the dimensionality reduction algorithms. In this section, we will try to see if we can describe visually how applying the above algorithms changed how both the Vehicle and PenDigits datasets are represented.

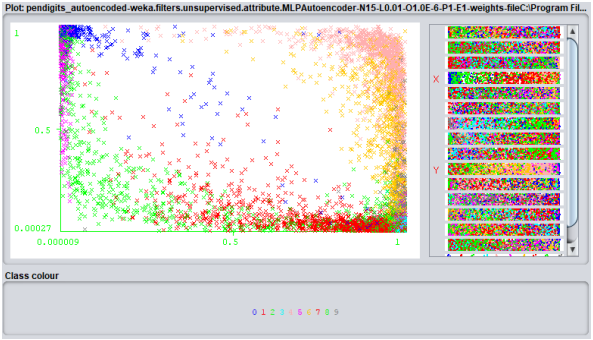
Using WEKA's Visualize feature, here are two handpicked graphs (selected for clearest cluster separation, PenDigits and Vehicle respectively) after applying PCA for each

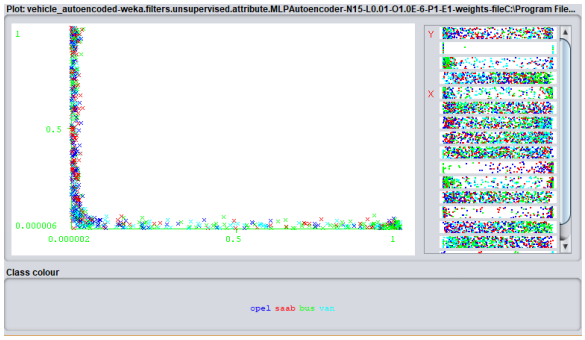




From the pictures above, we see that PCA (with explained variance @ 0.95) was able to separate out the labels pretty decently for the PenDigits dataset, note that there are two different colors of green although different to see in the picture. We can visually separate the different clusters, hinting toward good performance when we apply the clustering algorithms as locality of instances with similar labels is still maintained even after reducing the number of dimensions. Looking at the graph for the Vehicle dataset, we see primarily the mixture of the "saab" and "opel" labels (as expected from domain knowledge, authors of dataset specifically pointed out the similarity between those two labels), red and blue, with a separation from the "bus" and "van" labels. Overall, PCA performs poorly on the Vehicle dataset, probably due to lack of enough instances and due to the lack of distinct silhoutte images for the confusing labels, boding problems later on in the analysis when we get to clustering and neural networks. For PenDigits, running PCA required the use of the top 10 principal components to capture 95% variance (but only requiring the top three eigenvalues to capture roughly 70% of the variance and the other seven to capture the remaining 15%), whereas the algorithm only required the top 7 principal components to capture 95% variance in the Vehicle dataset (requiring the top two eigenvalues to capture roughly 70% of the variance and the other five to capture the remaining 15%). Ways to improve PCA, while balancing information loss, for both datasets would be add more overall instances for the Vehicle dataset and add more specific instances that help distinguish similar handwritten digits like "3" and "9" as well as "8" and "0" which may help separate those clusters that are overlapping.

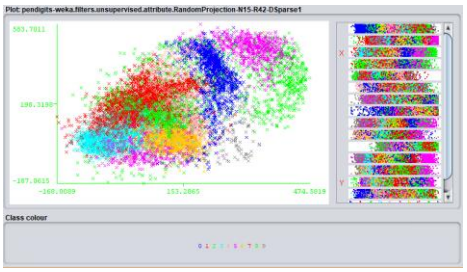
For AutoEncoding, I used a similar technique to select the pictures below, changing the number of hidden units until I was able to find one that resulted in good data separation:

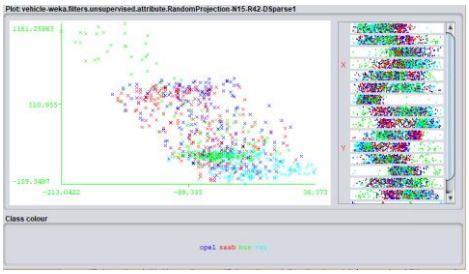




Autoencoding performed a lot better at maintaining label identity with respect to the Vehicle dataset while still reducing the number of dimensions to 15, the number of hidden units. We see a somewhat clear transition from red to blue to light blue to green. The features are distributed pretty sparsely (with the dots in the pictures above staying near the extremes of the graph) as expected by the autoencoding algorithm, due to the thresholding of the hidden units and the linear combination of non-linearity introduced by the sigmoid function. We see pretty good performance of autoencoding with respect to the PenDigits dataset, showing clear clusters while still maintaining the sparse nature expected from the algorithm. Ways to improve the algorithm would include the use of multiple hidden layers, each with decreasing number of hidden layers, to fully capture the underlying function.

Lastly, below are some pictures (PenDigits then Vehicle) gathered from multiple runs of Random Projections, with the objective being to retain graphs with the clearest clusters:



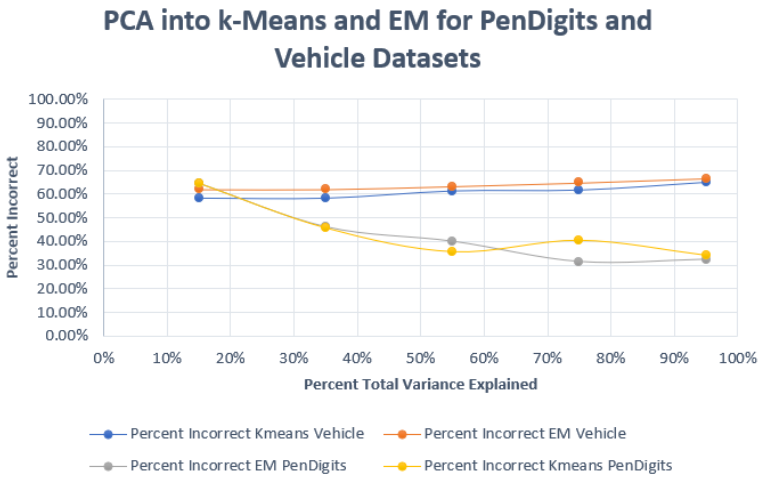


As in the name, Random Projections generated different clusters each time the algorithm was run (averages over several runs of metrics used in clustering were kept for the next component of the analysis). Overall, random projections did well for the PenDigits dataset, and decently well for the Vehicle dataset, able to clearly separate out the "van" and "bus" from the other two labels. Ways to improve random projections could be to run it more times or to change the seed value for randomness. Overall, across all dimensionality reduction algorithms, the PenDigits dataset was able to maintain the clearest clusters even with reduced dimensions compared to the Vehicle dataset, probably due to the larger number of clear, distinct instances. [Note: Improvements for these algorithms discussed in the next seection]

Dimensionality Reduction + Clustering:

In order to see the influence of the three dimensionality reduction algorithms (Principal Component Analysis [PCA], Auto-Encoding [EN], and Random Projections [RP]) on clustering, I applied each of the dimensionality reduction algorithms to both my Vehicle and PenDigits dataset and then feed the transformed data into both the *k*-Means and Expectation Maximization algorithms.

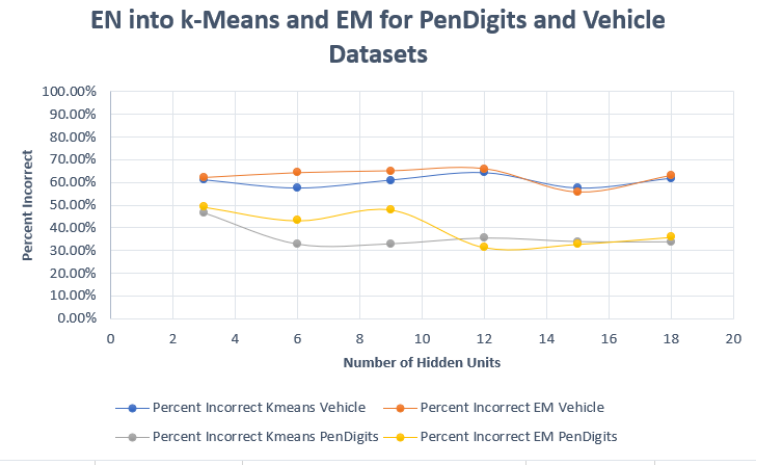
For PCA, I altered the amount of total variance explained by the principal components while the number of clusters at 4 for the Vehicle dataset and at 10 for the PenDigits dataset, as these were the optimal values from the clustering part of this analysis for both the *k*-Means and Expectation Maximization algorithms. I then recorded either within cluster sum of squares (for *k*-Means) or log likelihood (for EM), the number of iterations, and the percent of instances incorrectly clustered (using WEKA's Classes to Clusters Evaluation Metric). The results for running PCA then clustering on both datasets are shown below:



PCA Vehicle	**kMeans clusters = 4**		**EM clusters = 4**	
Variance Explained	Within cluster sum of squared errors	Iterations Kmeans	Log likelihood	Iterations EM
15%	2.961584914	16	-2.37048	100
35%	2.961584914	16	-2.37048	100
55%	17.91750018	25	-4.15586	48
75%	22.8598529	33	-5.14689	43
95%	104.3863977	15	-9.8387	100
PCA PenDigits	kMeans clusters = 10		EM clusters = 10	
Variance Explained	Within cluster sum of squared errors	Iterations Kmeans	Log likelihood	Iterations EM
15%	6.742975311	67	-1.89538	100
35%	92.40205995	90	-3.59451	81
55%	193.7425819	22	-5.00534	100
75%	550.4890682	25	-7.50481	41
95%	1194.242575	44	-11.81554	100

From the graph above, we see that for the PenDigits dataset, as the amount of total variance explained by the Principal Components increased, the percent incorrect decreased. This is identical for both the *k*-Means and EM algorithms. One simple explanation for this trend is that the larger the amount of variance explained, the more principal components were allowed to be presented in the transformed data leading to a better, compact representation for the original data. Continuing, the within cluster sum of squares (cohesion) and absolute value of the log likelihood value actually increased as the variance explained increased, showing how the more principal components used the more features were present in the transformed data allowing for greater heterogeneity amongst instances that were eventually clustered together. Moving on to the Vehicle dataset, from the graph, we see an unusual trend: as the amount of variance explained increased, the percent error also increased. This could be due to one of many sources of bias: reducing the dimensionality of the original dataset using orthogonal principal components is a worse representation of the original data, there is not enough data for the optimal principal components to be selected, or the clustering algorithms are having trouble finding appropriate cluster centroids for the transformed data. The within cluster sum of squares and log likelihood values for the Vehicle dataset exhibit similar trends as the PenDigits dataset.

For EN, I altered the number of hidden layers while keeping the number of clusters at 4 for the Vehicle dataset and at 10 for the PenDigits dataset, as these were the optimal values from the clustering part of this analysis for both the *k*-Means and Expectation Maximization algorithms. I then recorded the same metrics as I did for PCA. The results for running EN then clustering on both datasets are shown below:

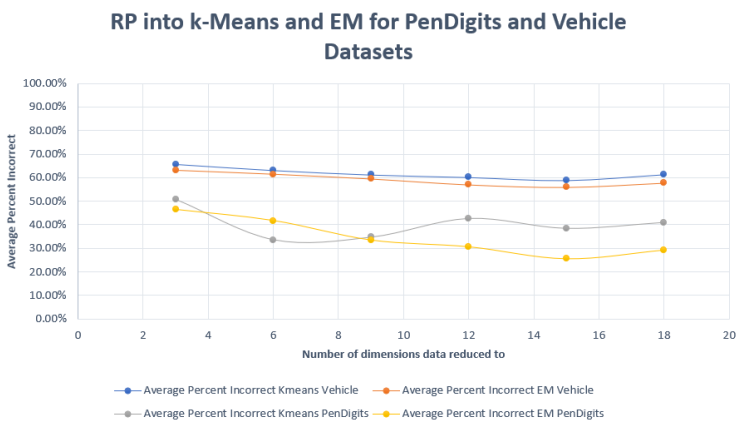


EN Vehicle	**kMeans clusters = 4**		**EM clusters = 4**	
Number of Hidden Units	Within cluster sum of squared errors	Iterations Kmeans	Log likelihood	Iterations EM
3	94.81048561	6	5.37573	34
6	362.087785	25	5.21334	51
9	513.0502418	10	6.11497	71
12	636.6322969	14	12.6469	35
15	775.103549	26	12.9024	3
18	843.3718628	36	20.78855	2
EN PenDigits	kMeans clusters = 10		EM clusters = 10	
Number of Hidden Units	Within cluster sum of squared errors	Iterations Kmeans	Log likelihood	Iterations EM
3	274.769095	23	6.58409	100
6	2979.551036	33	4.41631	75
9	5220.400835	37	4.81862	53
12	7196.705489	25	5.91114	50
15	8448.406289	36	6.17031	62
18	9573.483724	28	10.1204	49

Unlike the distinct decline in percent error we saw when applying PCA to the PenDigits dataset, when we apply EN to the PenDigits, we see similar values for percent error as the number of hidden units increase from 2 to 18. This simply tells us that changing the number of hidden units, effectively reducing the dimensions of the dataset to the number of hidden units, is not as influential as changing the variance explained (which makes sense as variance explained is a much more powerful influence over data representation) and it does not really matter how many units you use to represent the original data. This notion is spectacular as you can use a small number of hidden units to represent your

data and still result in comparable amounts of error. Similar to the PCA within cluster sum of squares and log likelihood, the values increase as the number of hidden units increase which makes sense as more features are created for transformed instances as the number of hidden units increase. Overall, EN performs as well as PCA does in providing a clear representation of the original data in a reduced number of dimensions. Moving on to the Vehicle dataset, we observe a similar trend expect the values for percent error are much higher than those for the PenDigits dataset, but comparable to the PCA values for the Vehicle dataset. Looking at both datasets, we see a decline in the number of EM iterations (similar to what we saw in the clustering portion of this analysis) and also observe that a hidden unit number of 15 generally results in the lowest percent error. One way to improve the EN + clustering is to add more and more hidden layers, reducing the number of hidden units each layer, until we reach a minimum percent error.

For RP, I altered the number of dimensions the data would be reduced to while keeping the number of clusters at 4 for the Vehicle dataset and at 10 for the PenDigits dataset, as these were the optimal values from the clustering part of this analysis for both the *k*-Means and Expectation Maximization algorithms. I then recorded the same metrics as I did for EN and PCA. The one key difference is that for RP, due to the inherent randomness, I ran each experiment multiple times and only recorded the average of all the recorded metrics. The results for running RP then clustering on both datasets are shown here:



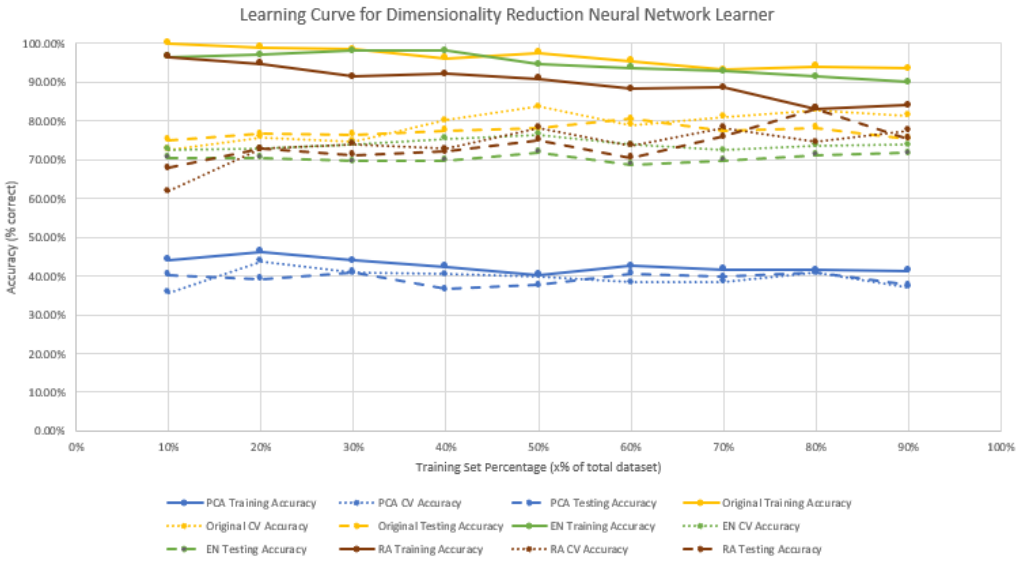
RP **Vehicle**	**kMeans clusters = 4**		**EM clusters = 4**	
Reduced to k Dimensions	Within cluster sum of squared errors	Iterations Kmeans	Log likelihood	Iterations EM
3	22.43381796	11	-15.41401	95
6	50.41666348	11	-30.14758	44
9	69.15182672	20	-44.94363	20
12	95.74593395	22	-59.06252	65
15	152.9707382	15	-89.68895	20
18	124.7645164	28	-73.9016	100
RP **PenDigits**	kMeans clusters = 10		EM clusters = 10	
Reduced to k Dimensions	Within cluster sum of squared errors	Iterations Kmeans	Log likelihood	Iterations EM
3	165.3916144	43	-17.27749	52
6	485.6651714	31	-33.64098	39
9	829.5438504	38	-47.50806	8
12	1174.569398	26	-61.82923	14
15	2238.811082	66	-96.20884	10
18	1581.268394	23	-78.47747	13

Similar to EN, for the PenDigits dataset, the percent error did not change that much as the number of dimensions the data was reduced to changed. Only for EM did the percent error value at 18 dimensions appear clearly lower than the value at 2 dimensions. This observation opens the way for one obvious conclusion: it does not significantly matter how many dimensions you utilize to convey your data. This is powerful as we can effectively reduce our data to very low dimensions and still have comparable values for percent error. The Vehicle dataset outputted similar trends as the PenDigits dataset but much smaller in magnitude (very small, steady decline in percent error). Both datasets, similar to PCA and EN, resulted in increases in the within cluster sum of squared errors and log likelihood values and these can be attributed to the increase in the number of features as the number of dimensions increases which results in clusters in higher dimensions being more heterogenous.

Overall, running dimensionality reduction algorithm did not result in drastically higher percent error values for either the PenDigits or the Vehicle dataset. What this tells us is that we can effectively reduce the dimensions of our datasets and apply clustering without worrying about losing information. Not only does this result in faster algorithm times, but it also allows us to visualize the data and the clusters easier in addition to observing which features/dimensions of the original dataset matter more when it comes to deciding which clusters specific instances belong to. Ways to improve this portion of the analysis were discussed in the initial clustering analysis and in the dimensionality reduction algorithms analysis. By adding both of those improvements, I believe dimensionality reduction into clustering would also improve.

Dimensionality Reduction + Neural Network Learner

One way to truly compare the influence of reducing dimensionality of your dataset is to pass in the transformed instances into a neural network learner. For this part and for the rest of the analysis, I used the Vehicle dataset as it was able to train quickly, due to the relatively small number of instances, and because during Assignment 1, the vanilla neural network was not able to very effectively separate the data (around ~75% accuracy) and I wanted to see if reducing the dimensionality would increase that accuracy. To perform this portion of the analysis, I first ran the Vehicle dataset through each of the three dimensionality reduction algorithms fixing their hyperparameters at the values that exhibited the smallest percent error during the dimensionality + clustering portion of the analysis (these values are highlighted in the tables above and are also repeated here: for PCA: variance explained held @ 0.15, for EN: number of hidden units held @ 15, and for RP: the number of dimensions to reduce to held @ 15). I then split up the transformed data into training and testing sets of various sizes. After doing that, I ran the transformed datasets into WEKA's "MultilayerPerception" algorithm keeping learning rate at 0.3, momentum at 0.2, and the number of epochs at 500. I also went back and reran the neural network learner for the original, untransformed Vehicle dataset at various training and test set sizes to create the following learning curves:



Right off the bat, we can see several distinct trends present in the learning curve graph above. First and foremost, out of the four different inputs to the neural network learner, the PCA inputs performed the worst. With an average classification accuracy of around 45%, the PCA transformed instances also did not show much improvement in testing and cross-validation (CV) accuracy as the size of the training set increased, nor did the training accuracy decline that much. However, if we look back at the performance just clustering and of PCA + clustering, we see that in the Vehicle dataset, the error was still around 60% – 70% for both techniques, and now when using PCA + neural network we see an error of around 55% which is a decent improvement. One possible explanation for this could be that the neural network is able to distinguish the underlying non-linear differences between the transformed input data whereas clustering simply uses Euclidian distances to group similar instances.

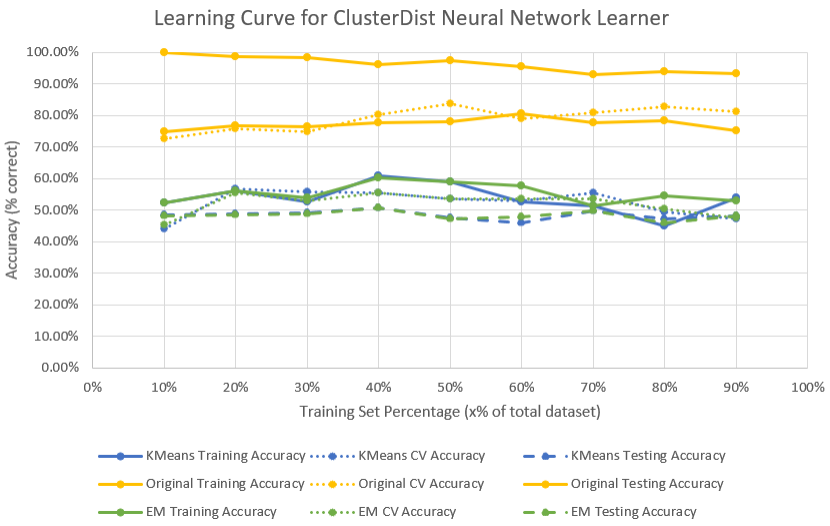
In an ideal situation, as your training set size increases, your testing accuracy initially starts off near 100% as your learning simply memorizes the inputs and then gradually decreases due to variation in your input data. Also, your testing and CV accuracies tend to increase to some maximum as the training set size increases, as we control the amount of memorization and our model becomes more robust, and then begin to decline later on, as we now risk overfitting with so many training examples and so few testing examples. In the learning curves above, we see this trend with the original, untransformed data, the EN-transformed data, and the RP-transformed data. Looking specifically at the EN-transformed data, we see very similar trends with the original data with respect to the training accuracy, but the testing and CV accuracies are distinctly lower. This is simply due to the loss of information we accept when performing autoencoding. If we take our original data, pass it through a neural network in hopes of compressing it and then pass it through another neural network in hopes of classifying instances, then it is not unreasonable that we lose accuracy along the way. The good thing is that the accuracy loss is very little, roughly 8% loss, and as a result, autoencoding the data first could be a plausible alternative to simply running the neural network with the original data if we are concerned with algorithm runtime as the number of instances increase. Looking at the RP-transformed data, we see similarities with the original data with respect training accuracy, but unlike EN, RP tends to drop training accuracy more sharply as the training set size approaches 90%. The tradeoff to this is the better testing and CV accuracy of RP when compared to EN. What the RP learning curve basically tell us is that we can project our Vehicle dataset into 15 dimensions and have a similar level of accuracy regardless of the training set size.

Overall, the original, untransformed neural network performed the best, followed by the RP and EN-transformed neural network, and lastly by the PCA-transformed neural network. However, each dimensionality reduction transformation resulted in better neural network performance than clustering performance. Ways to improve the neural network include experimenting with different learning rates, momentums, and epochs as we did with Assignment 1 along with adding more instances as the Vehicle dataset only contains 846 instances.

Clustering + Neural Network Learner

What if , instead of applying dimensionality reduction algorithms to the data, we used generated cluster centroids (from the *k*-Means and EM algorithms) to transform our input data

and then feed those transformed instances into the neural network learner? What kind of trends would we see in learning rate and how will those trends compare to the original, untransformed data? To answer this question, I took every instance x_i from the Vehicle dataset and calculated the Euclidian distance from x_i to each of the four clusters generated by *k*-Means and the four clusters generated by EM. I then created two new datasets, both with the exact same number of instances as the Vehicle dataset, but with transformed x_i's such that $x_{i,\ Kmeans}$ = {|distance to cluster1|, |distance to cluster2|, |distance to cluster3|, and |distance to cluster4|, original label} and $x_{i,\ EM}$ that followed the same formula as *k*-Means except only the cluster mean was used. I then saved $x_{i,\ Kmeans}$'s to one dataset and the $x_{i,\ EM}$'s to another dataset and run both datasets through a neural network learner at various training set sizes, resulting in the following learning curve:



Looking at the learning curves, we see similar performances between the *k*-Means cluster instances and the EM cluster-based instances. Performing roughly around 50%-60%, we see an improvement of training, testing, and CV accuracy if compared to the PCA neural-network from the dimensionality reduction + neural network learner portion of the analysis, but does not match the accuracy of the EN and RP neural networks. As we created entirely new instances composed of distances from each cluster center, it is not unreasonable to see changes in accuracies if we altered the cluster centers. Perhaps one way to improve upon this neural network is to run the clustering algorithms with either more original instances or for a longer period of iterations. Additionally, we could add more hidden layers to see if there is any non-linearity that is not being captured by just having one hidden layer with a certain number of hidden units. Lastly, an important component of the analysis we must consider is the loss of information tradeoff we accepted when we transformed the original data with 18 attributes to new data with just 4 attributes. As this can essentially be treated as dimensionality reduction, then the neural network learner did fairly well to capture the underlying function, suffering only about 15%-20% accuracy loss.

Conclusion

Overall, I gained a deeper understanding of clustering and dimensionality reduction algorithms over the course of this assignment along with how we can combine unsupervised techniques with supervised techniques. As with all experiments in Machine Learning, more data would lead to better approximations of the true performance values of these algorithms. However, by performing these analyses, we also learn how tweaking hyperparameters can lead to performance boosts and use that knowledge along with domain knowledge to create a better representation of the problem at hand.