Unsupervised Learning and Dimensionality Reduction

Tiffany Xu

Dataset Description

The income dataset contains 14 attributes from a census survey (e.g. age, occupation, marital status, country) that are used to predict if a person's income is below or above $50,000. In order to run the algorithms in a reasonable amount of time, the data was cut in half, to a size of 16,280 instances. This is also the dataset that will be used to rerun the neural network in the latter part of the project, and the structure of the neural network remains the same from the previous projects with 11 nodes in the hidden layer.

The letter recognition dataset contains 20,000 instances of letters (A-Z, capitalized) with 16 attributes describing the physical properties of each letter, such as xbar, width, ybox, etc. The data was based on 20 different fonts, and each letter in a font was furthermore randomly distorted to produce the 20,000 instances.

I predict a greater improvement with dimensionality reduction on the income dataset because there are many attributes that seem to be redundant (e.g. marital status vs. relationship or education.num vs. education), some attributes that are more kurtotic that others (i.e. fnlwgt), and some attributes that probably matter more than others in determining the classification (e.g. education is probably a better predictor than relationship). It is more difficult to determine which attributes are important in the letter dataset since they are all measures of some physical attribute, and most attributes are normally distributed. It will be interesting to see the effects of unsupervised learning and dimension reduction on these two different datasets.

For all the clustering algorithms, I ignored the class attribute because one of the goals of this project is to see how clustering affects classification. If class attributes are included, clusters could be based on the class label, which would trivialize the classification problem.

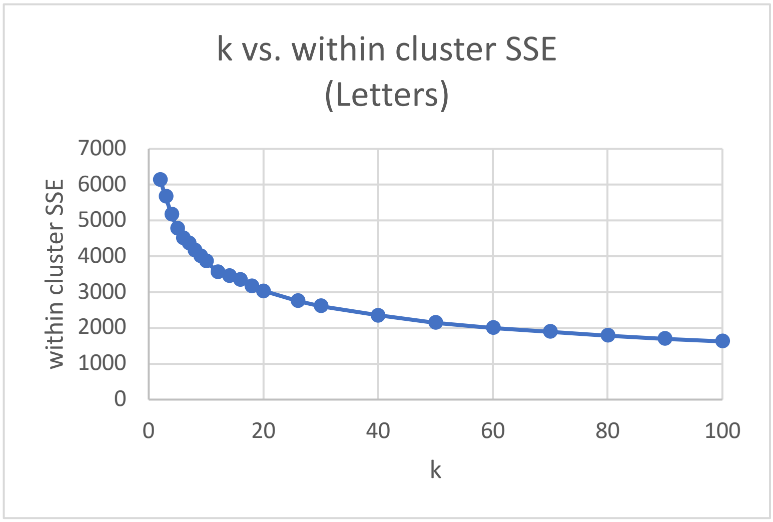
k-Means Clustering

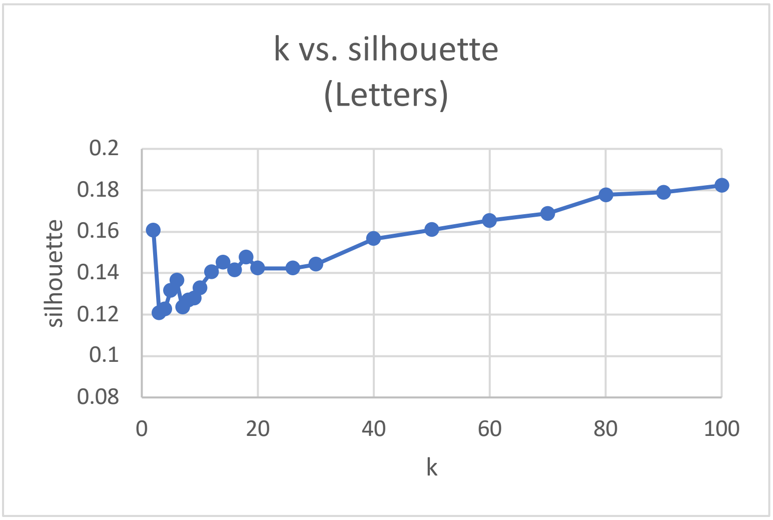
k-Means clustering is rather intuitive; the algorithm chooses k random initial points and the instances closest to each of these points are considered part of that cluster. The center of each cluster is recomputed based on the mean of the points in that cluster, and once again the points closest to this new center are considered part of that cluster. This continues until convergence, when the center point does not change anymore.

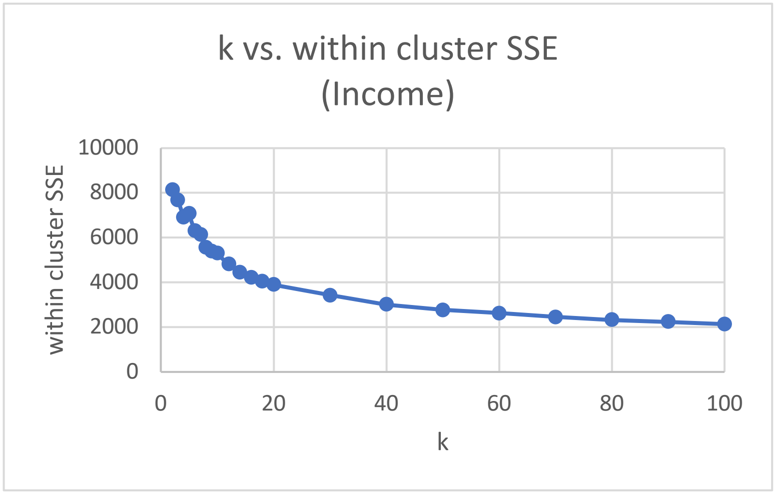
To determine which distance metric to use, both Euclidean and Manhattan distances were used to run an initial k-means clustering trial. Euclidean distance yielded a smaller within cluster sum of squared errors and was thus chosen to be the distance metric.

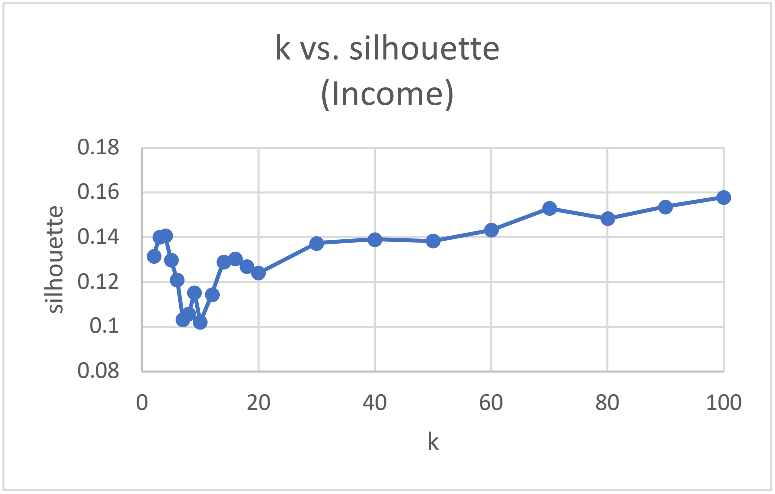
The hyperparameter for k-means is the value for k, which represents the number of clusters, and the goal is to minimize the variation with in each cluster. The graph below shows the k value graphed against its corresponding within cluster sum of squared errors. The within cluster SSE is a measure of variability in each cluster and is proportional to the number of instances (i.e. if there are more instances in the dataset, the errors are usually higher). I initially hoped to use the elbow method to determine the optimal k value. There is no obvious elbow, however, so I graphed the k value against its silhouette value, a more mathematically based interpretation of consistency for clustering. As a compromise between the elbow, the silhouette, and the fact that the letter dataset is classified into 26 classes, I chose 26 as the optimal k value. The silhouette values before k = 26 are inconsistent and jump up and down, and k = 26 still lies within the elbow of the graph. Another reason for choosing the smallest consistent value of k is to avoid overfitting.

For the income dataset, the k value at which the elbow begins to smooth out, as well as the k value after which the silhouette values remain more consistent is 30. Thus, 30 was chosen to be the optimal k value for the income dataset. This could mean there are 30 general types of people who took the census for the income dataset, since the clusters were formed based on census attributes.







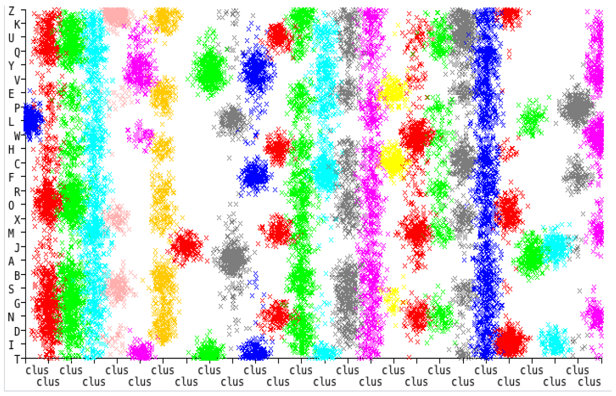


Expectation Maximization Clustering

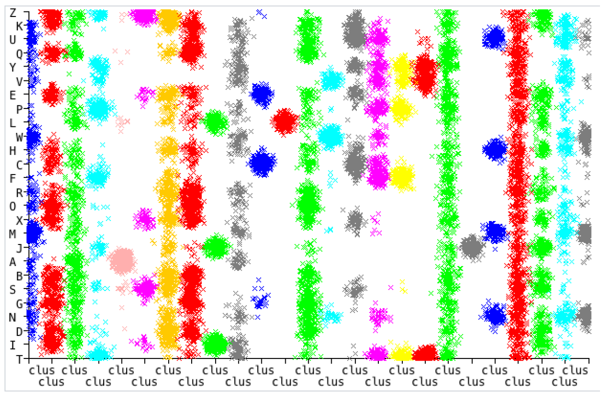
Expectation maximization is not as cut and dry as k-Means. An instance, instead of being exactly in one cluster, is in each of the k clusters with a certain probability. This is also known as soft clustering, and is useful in cases where, for example, a point might be exactly in between two clusters, so that instance can be 50% in cluster A and 50% in cluster B.

WEKA is able to use cross-validation to find it's believed optimal k value. We will be using log-likelihoods to compare the different k values. Log-likelihood is used in practice as a measure of likelihood because it is concave, and concave functions are easier to maximize than non-concave functions. For the letter dataset, this feature chose k = 25 with a log likelihood of -28.83064. I then ran EM with k = 26, the optimal k value found from k-Means, which yielded a log likelihood of -28.54951. To determine the best k value for EM, we want to maximize the log likelihood, so I chose k = 26 to be the best k value for expectation maximization on the letter dataset.

For the income dataset, the WEKA cross-validation determined k value was 7 with a log likelihood of -39.12466. When running EM with a k value of 30, the k value found in k-Means, the log likelihood was -42.94008. Because 7 yields a better log likelihood, and is less prone to overfitting, due to both is smaller number and the fact that it was determined with cross-validation, 7 is determined to be the optimal k value for expectation maximization on the income dataset.

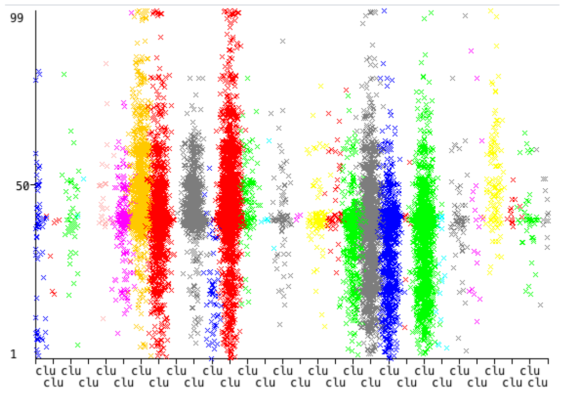


26 clusters vs. letter

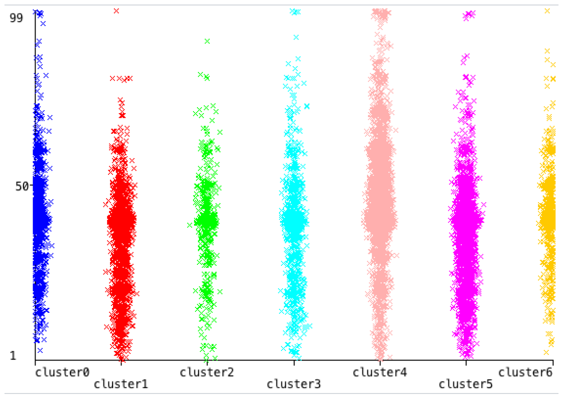


25 clusters vs. letter

Above is the visualization of clusters for the letter dataset and belong is the visualization of clusters for the income dataset. Because neither of these datasets have less than or equal to 2 attributes, I just chose an attribute to map against the clusters to see a small perspective of the clusters and how they compare with different k values. This comparison should still be valid since the only thing changing in the visualization is the k value, but the attributes shown are the same. For the letter clustering, there is not much difference between the 26 clusters and 25 clusters. 26 clusters seem less concentrated around the center, but this is slightly due to the jitter used to clarify the visualization. For the income dataset, the 7 clusters have more instances in each cluster because of the smaller number of clusters, while the 30 clusters is more dispersed with some clusters containing relatively few instances.



30 clusters vs. hours per week



7 clusters vs. hours per week

For the rest of the analysis, a k value of 26 will be used for the letter dataset for both k-Means and expectation maximization. For the income dataset, a k value of 30 will be used for k means and a value of 7 will be used to expectation maximization. This will allow for consistency in comparison among different types of transformations.

Principal Components Analysis (PCA)

Principal components analysis is a feature transformation algorithm that allows us to linearly transform and combine the attributes of a dataset in a way that extracts the useful parts of each attribute while disposing of the remaining useless parts. Unlike feature selection algorithms that only pick and choose which attributes to keep, and thus losing whatever potentially useful information the discarded attributes might contain, PCA allows us to keep only those useful parts. Not only do we maintain the useful information each attribute brings, we are also able to reduce the number of attributes, alleviating the curse of dimensionality. This is accomplished by way of finding directions of maximal variance among the instances and ranking the importance of the attributes by their eigenvalues (higher is better).

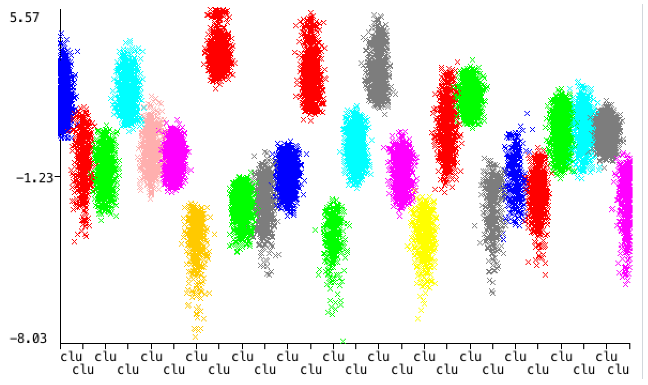
The table below summarizes the results with different runs of the PCA. The variance covered was the hyperparameter that correspondingly changed the number of resulting attributes. For example, if the variance covered is 0.95, that means we want to add attributes until 95% of the overall variability is reached. Thus, it makes sense that the higher the variance covered, the more attributes are needed to cover it.

Dataset	Clustering Algorithm	Variance Covered	# of result attributes	Log Likelihood	Time Elapsed	Within cluster SSE
letter	k-Means	0.95	31	-	3.25s	5984.387508790959
letter	k-Means	0.75	21	-	1.34s	4160.894233060325
letter	k-Means	0.55	14	-	0.71s	1997.6794160252077
letter	k-Means	0.35	7	-	2.93s	788.7931386958101
letter	**k-Means**	**0.15**	**3**	**-**	**1.43s**	**163.6189682633013**
letter	**EM**	**0.95**	**31**	**15.15451**	**19.04s**	**-**
letter	EM	0.75	21	6.13738	14.9s	-
letter	EM	0.55	14	-11.76922	29.11s	-
letter	EM	0.35	7	-9.55268	24.97s	-
letter	EM	0.15	3	-5.34671	19.45s	-
income	k-Means	0.95	14	-	2.33s	1254.344316730413
income	k-Means	0.75	10	-	2.82s	613.3212100880123
income	k-Means	0.55	7	-	3.24s	360.3638473002857
income	k-Means	0.35	5	-	2.02s	227.24877381665584
income	**k-Means**	**0.15**	**3**	**-**	**1.18s**	**41.05324845147137**
income	EM	0.95	14	-12.3554	26.15s	-
income	EM	0.75	10	-7.19885	23.39s	-
income	EM	0.55	7	-3.62489	15.39s	-
income	EM	0.35	5	-2.12947	17.36s	-
income	**EM**	**0.15**	**3**	**1.69546**	**20.55s**	**-**

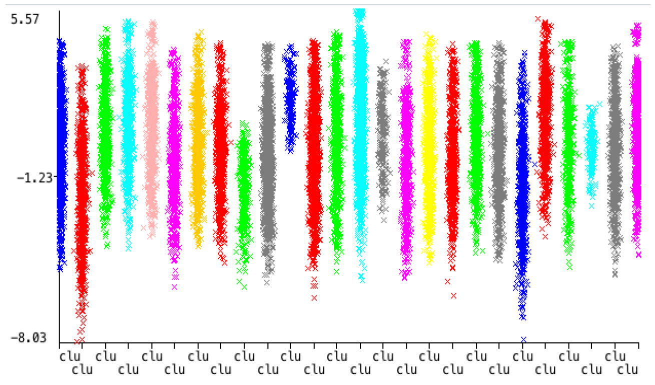
For the letter dataset, k-Means did best with the lowest variance covered of 0.15. Interestingly for EM, the highest variance covered at 0.95 did best. Since EM is a soft clustering algorithm, having more information allows it to better assign distributions for each instance. It is important to note that the number of attributes for 0.95 variance covered also increased the number of resulting attributes because the number of attributes relative to the number of instances is very small, so the features were projected into more dimensions to ensure an epsilon safe projection. For the income dataset, both k-Means and EM performed best with a variance covered of 0.15. Perhaps the overall variance in the income dataset was not as widespread as the variance in the letter dataset, so even covering a small percentage of variance was capable of clustering that data in a centralized way. These results are all much better than the clustered data found without principal components analysis; PCA does seem to have helped clustering so that each instance is a shorter distance away from the cluster center, as well as increasing the log likelihood for expectation maximization. The best results are bolded in the table above.

There is also a general negative trend between the number of attributes and the running time. The lesser attributes there are, the less complex the analysis, since the clusterings would be dependent on less dimensions. Additionally, expectation maximization takes much longer to run than k-Means, which usually finishes in an instant. This is because instead of deciding which single cluster a data point fits in, expectation maximization calculates its probability of coming from every single one of the clusters, which computationally takes longer.

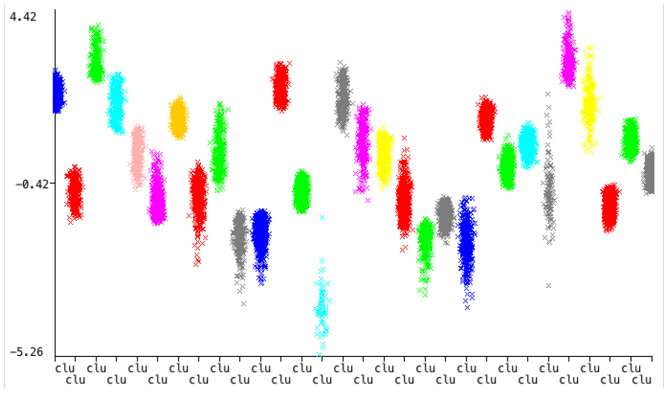
Below are the clustering visualizations after running PCA on the data. For the letter dataset, the k-means clusters are quite dissimilar from the EM clusters, while the k-Means and EM clusters for the income dataset are very similar. These clusters do not look similar to the clusters produced without PCA in the previous section, although it is hard to compare these cluster visualizations since the attributes on the axis are different.



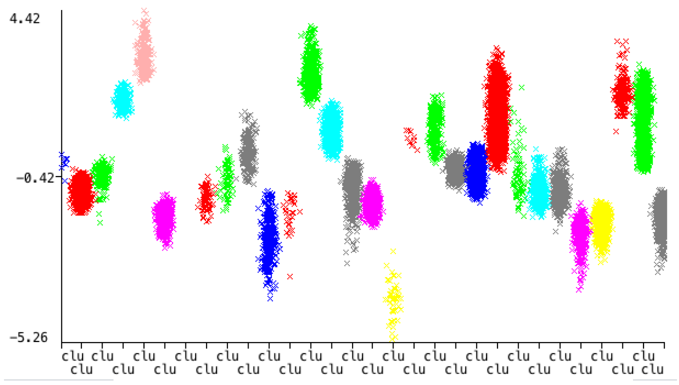
cluster vs. -0.432 width-0.418 x-box-0.415 onpix-0.379 high-0.377 y-box for k-Means



cluster vs. -0.432 width-0.418 x-box-0.415 onpix-0.379 high-0.377 y-box for EM



cluster vs. 0.527relationship-0.472sex-0.383hours.per.week+0.317marital.status-0.285age for k-Means

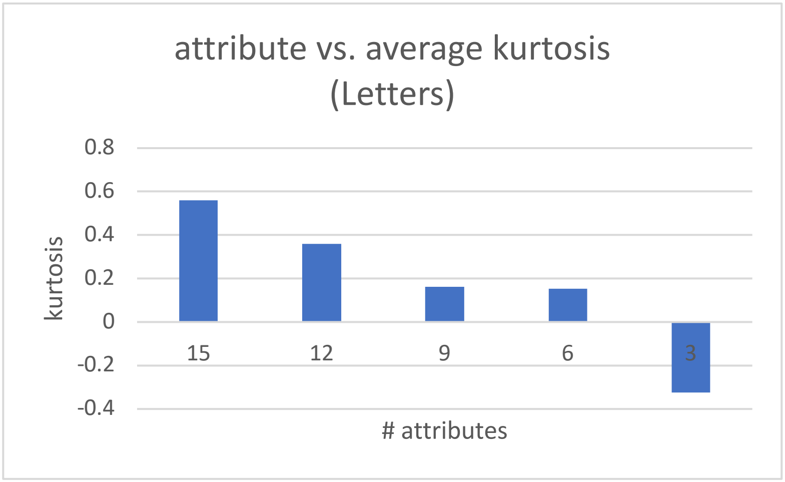


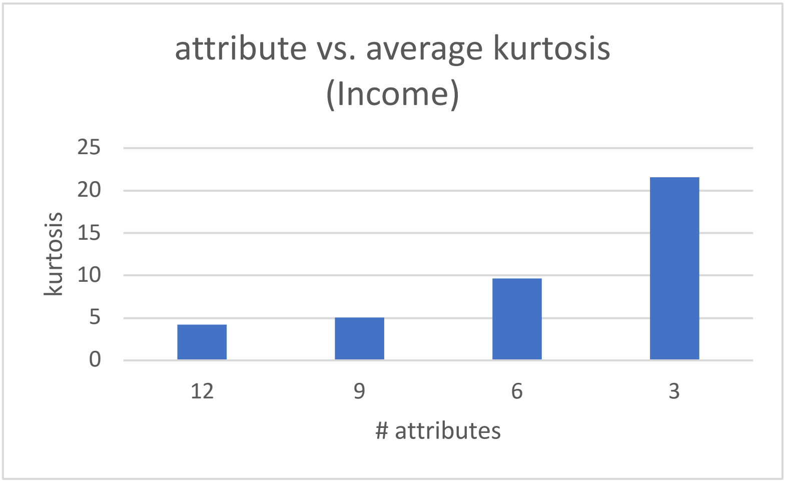
cluster vs. 0.527relationship-0.472sex-0.383hours.per.week+0.317marital.status-0.285age for EM

Independent Component Analysis (ICA)

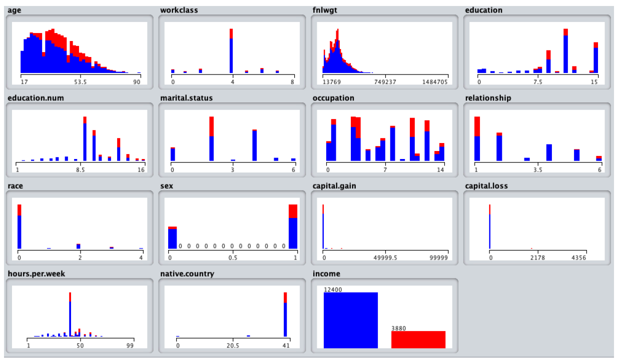
Independent component analysis, as the name implies, works best on data that is made up of mutually independent and non-Gaussian components that were linearly combined in some way. If so, ICA can separate these components, which is especially useful for blind source separation problems. This is done by maximizing the mutual independence among the new attributes, but also maximizing the mutual information between the original attributes and the new transformed attributes, to retain as much information as possible.

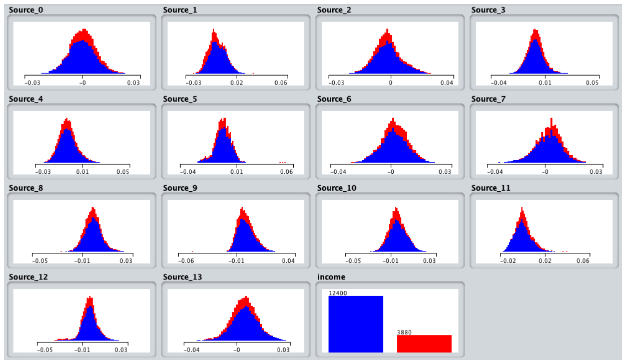
Because ICA assumes the data is non-Gaussian, the average kurtosis on our transformed features should not equal three, the kurtosis of the normal distribution. For hyperparameters, I altered the number of attributes in the transformed data, and calculated the average kurtosis. Overall, the kurtosis of the income dataset is much bigger than the kurtosis of letters dataset, but most projections have a kurtosis far from 3, except for in the income dataset with 12 and 9 attributes.



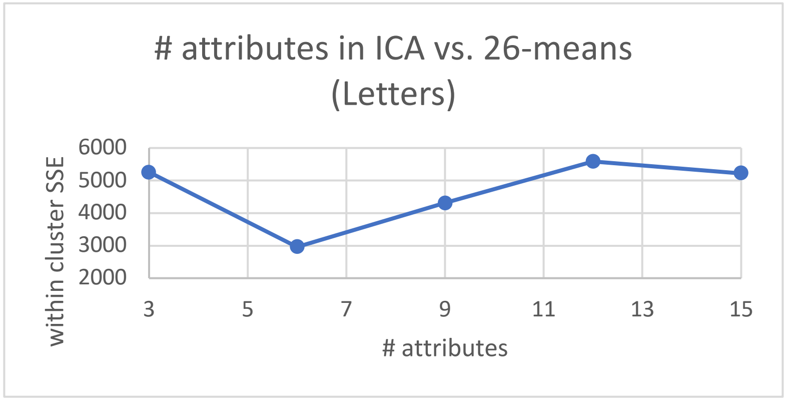


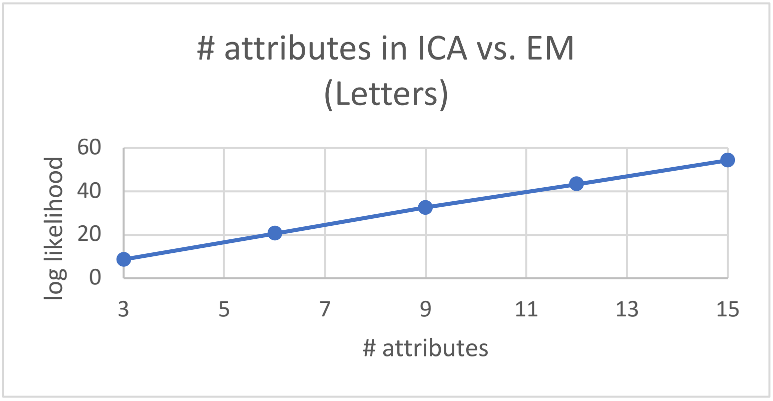
Below is a visualization of the distribution of the data within each attribute before (left) ICA and after ICA (right). The attributes become more focused around the peaks, changing the kurtosis for each attribute.

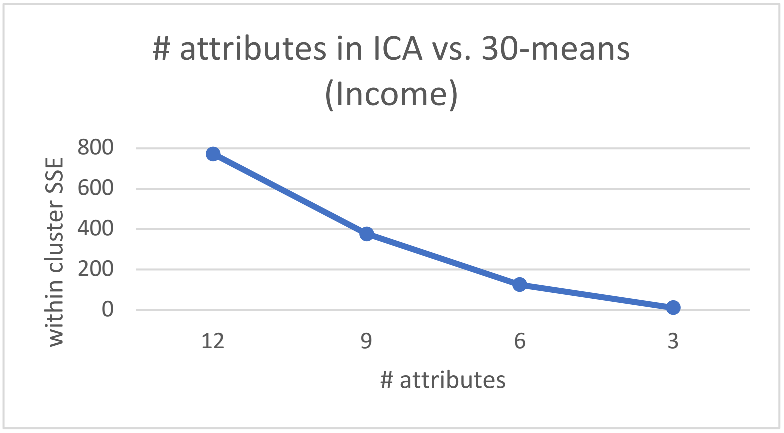


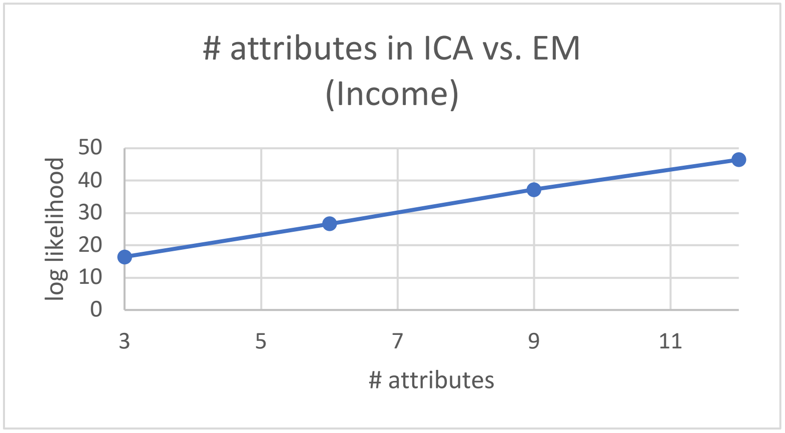


Because most of the ICA transformations have a kurtosis far from 3, the transformed features seem to be mutually independent. Running this transformed data against clustering again, we see the results in the graphs below. The lowest within cluster sum of squared errors for the letter dataset is at 6 attributes. For the income dataset, it is at 3 attributes. For both datasets, the expectation maximization had an exact linear relationship between the number of attributes and log likelihood, making it harder to determine the best k.





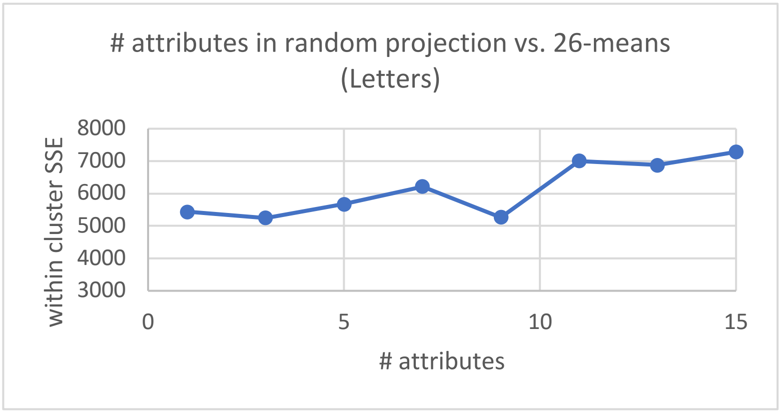


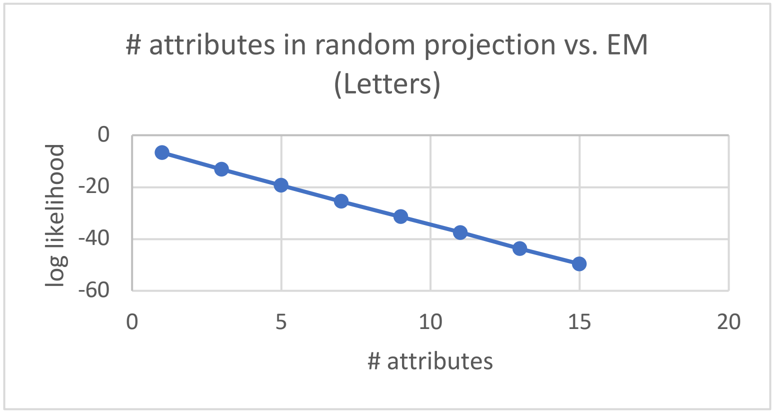


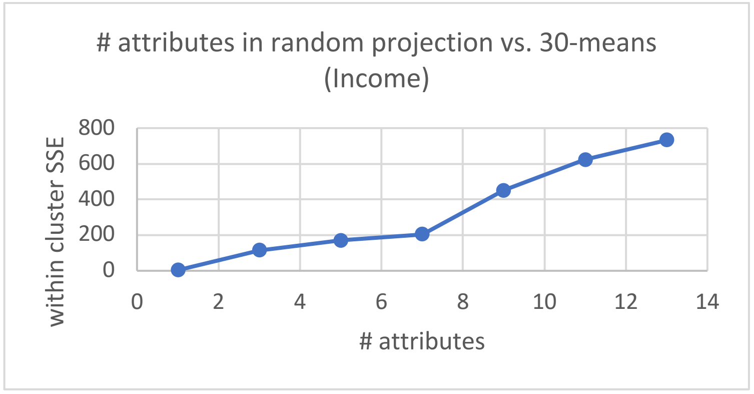
The log likelihoods for both datasets are significantly larger than when run with PCA. The average log likelihood for the letters was -1.075344, and -4.72263 for the income in PCA. For ICA, these values are 31.895302 and 31.6884725, respectively. Additionally, the kurtosis was seen to be far from 3, with the letter dataset a little bit more than the income dataset, and this result is paralleled in the slightly greater average log likelihood for the letter dataset. ICA is thus predicted to do better in the neural network with the EM cluster added as an attribute compared to the PCA EM cluster.

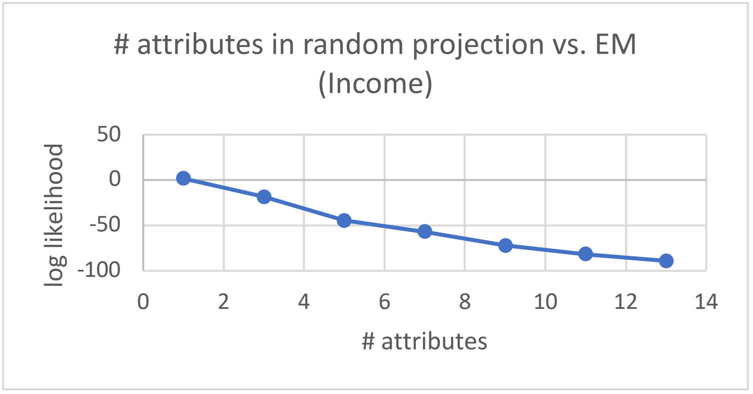
Random Projections

Random projections generate random directions onto which the data is projected. This method generally works well if it is used prior to doing classification. Additionally, the best advantage of random projection is that it is cheap, easy, and fast. Once again, the hyperparameter I altered was the number of dimensions to project into. For the letters dataset, the optimal k value is at 9 while it is 7 for the income dataset.









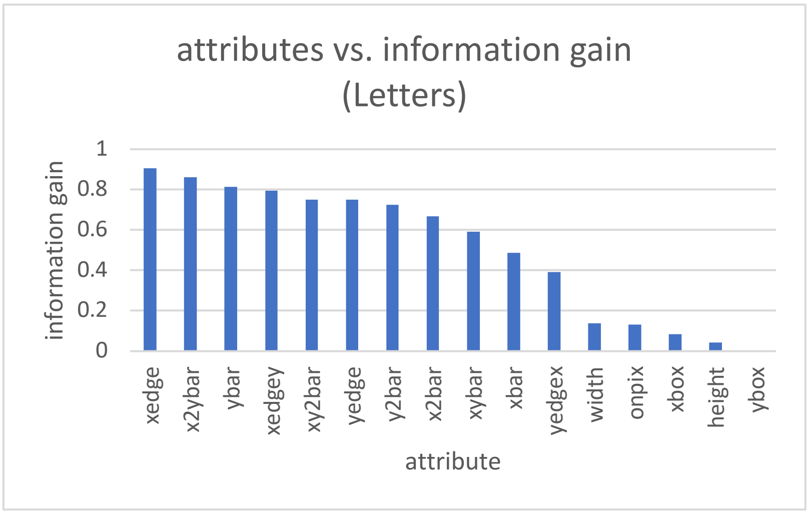
Initially, I ran random projection on the letter dataset with a seed of 42, and there was no trend in the data. When running random projections again with a seed of 3, though, we get the results seen above. Similar to randomized hill climbing, it is useful to rerun the projections with different seeds, paralleling random restarts to obtain better results.

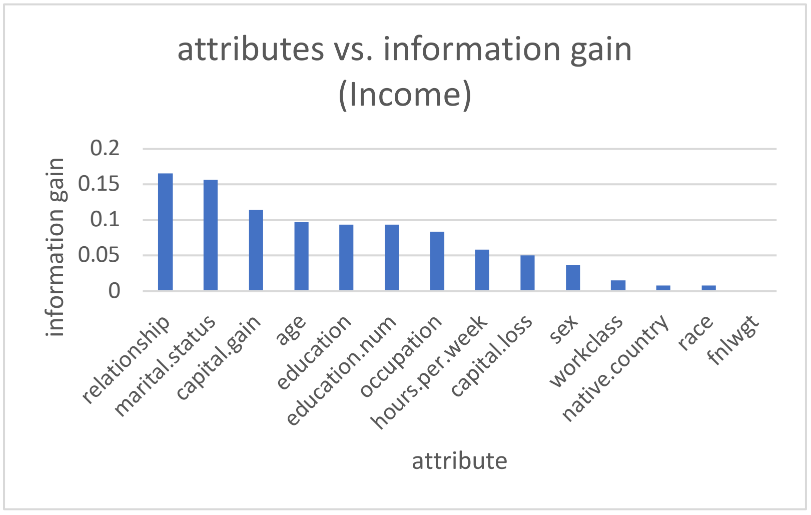
For the income dataset, there is a slight elbow at 5, but the graph for the letters is perfectly linear. It is difficult to determine the optimal number of attributes for expectation maximization, similar to the case in independent component analysis in the previous section, but instead of a positive relationship, this one has a negative relationship. Given the opposite trends for ICA and random projection in expectation maximization, I can only say that I should have perhaps analyzed these algorithms a different way, by choosing less variations in the number of attributes, but also testing out different k values for those specific dimensions. This should yield better results that would hopefully have had either an elbow or an eventual convergence.

Information Gain

Information gain, unlike the previous three feature transformations, is a feature selection algorithm. This is the same metric used to determine which attribute to split on in ID3 decision trees. Information gain is based on information theory, so we want the attributes that cause the greatest reduction in entropy after using it.

For the letter dataset, ybox had no information gain, so this attribute is removed. ybox represents the vertical grid location, and it's interesting that this attribute provides no information, but xbox (horizontal grid location) does. It seems that the vertical positions of letters do not distinguish it from other letters as much as the horizontal position does. For the income dataset, the fnlwgt attribute is removed. It makes sense that fnlwgt provided no additional information because this attribute was added based on other attributes to represent the believed number of people that the instance represents in the population. While fnlwgt could be useful in predicting other classes, like race or country, it does not provide any information in classifying income.





For this analysis, I only removed the one attribute that had zero information again, but it could be argued that the last 5 attributes in letters dataset (width to ybox), and the last 4 attributes in the income dataset (workclass to fnlwgt) could be removed, since these attributes provide much less information gain than the others. However, the attributes that could be removed but which I didn't still contain a nominal amount of information gain, so I decided to keep them.

Neural Network with Dimensionality Reduced Data (Income)

Dimensionality Reduction Algorithm	Training Accuracy %
None	85.7678
PCA	80.3993
ICA	81.5971
RP	76.9656
IG	85.7064

Unfortunately, the dimensionality reduced data all did worse than the original neural network run. Information gain came in close, but information gain also was changed the least by the feature transformation, only getting rid of a single attribute, so it's not surprising that information gain was the most similar to the original neural network accuracy. Random projection did much worse than the others, but perhaps running another random projection with a different seed could have yielded better accuracy.

Neural Network with Clustering as Features

To analyze the possible benefits that clustering can add to neural network classification, I added an attribute that represents the clustering of the data. The cluster attribute is tested against the neural network both alone as the only attribute (cluster only) and added in addition to the original attributes (cluster appended).

We can see that when the cluster attribute is used alone, the classification accuracy is horrible; the clustering attribute without dimensionality reduction and the one using information gain, though, do relatively better than the rest for both k-Means and EM.

However, appending the cluster as an attribute does improve the classification accuracy a considerable amount. From the previous section we see that the unaltered neural network accuracy is 85.7678%. Every single appended cluster except for one, improves that accuracy, as shown in the third and fifth columns. The added EM cluster attribute with no dimensionality reduction algorithm is the only one that had a worse accuracy at 85.2518%. This is a very small difference in accuracy, and may be a result of the additional information brought from the added dimension not outweighing the curse of dimensionality cons.

Interestingly, the overall best reduction algorithm was also the simplest one—information gain (last row). This algorithm with its cluster appended as an additional attribute yielded the highest accuracies for both k-Means and EM. However, the overall best accuracy is for k-Means. This earns another point for Occam's Razor; the simplest clustering and simplest dimensionality reduction yielded the best improvement on classification accuracy. This perhaps also means the attributes (other than the added cluster) were already sufficient in predicting the classification without having to transform in any way. This is also due to the fact that my datasets had a lot of instances relative to the dimensions, so dimensionality reduction did not help as much as initially predicted, especially since the curse of dimensionality was not that much of an issue anyways.

Now knowing that information gain was the best, it would be interesting to rerun these tests with even more attributes removed. Recall that only the least information gain attribute was removed from the income dataset, but in the information gain section I wrote about how 3 additional attributes could be removed too (race, native.country, workclass). These attributes had a very small information gain (but still greater than 0), and I wonder if removing these attributes would result in a better accuracy or worse considering that there was still a little bit of information gain. I did rerun the neural network with those attributes removed, and the accuracy did indeed increase, but only for the k-means cluster.

Dimensionality Reduction Algorithm	**k-Means Training Accuracy % (cluster only)**	**k-Means Training Accuracy % (cluster appended)**	**EM Training Accuracy % (cluster only)**	**EM Training Accuracy % (cluster appended)**
None	11.9349	94.9201	22.9545	85.2518
PCA	6.9226	88.4582	8.0958	88.6916
ICA	7.5921	87.3096	6.1609	88.3292
RP	10.9767	89.0356	9.6929	86.5479
IG	11.9349	95.1966	27.7703	91.855

Accuracy for information gain with additional attributes removed

Dimensionality Reduction Algorithm	**k-Means Training Accuracy % (cluster appended)**	**EM Training Accuracy % (cluster appended)**
IG	97.1314	89.7297

Future Work

If I were to redo this project, I would add a few more analysis, such as changing k after applying the dimensionality reduction algorithms. I did not initially do this because I thought it would be too many variations among the two datasets, two different clusterings, the four different dimensionality reduction algorithms, and the hyperparameters within those reduction algorithms to fit into this report. However, it is important to acknowledge that the best k value for k-Means and EM is likely not to be the best k value for k-Means and EM after a specific dimensionality reduction is processed. This could also help fix some issues in determining the best hyperparameters for expectation maximization. For most of the analysis with EM, the graph were perfectly linear—something improbable and incorrect. Perhaps changing the k values would reveal more useful information on this front.

I would also redo all of the analysis, but with the class attribute remaining in the dataset. I realize now that this doesn't necessarily trivialize the classification problem because the clustering problem itself may or may not use the class attribute to form clusters, which are not by default created around just the class value. This would also allow me to see how similar to clusters are to the labels of the instances in that cluster—if it is, this would be a great improvement to the classification problem, especially since clustering generally is much faster than running the neural network.

I would've also liked to have done more research on how to better compare projections. For example, I wasn't sure how to compare the normal k-means clustering against the k-means clustering after principal components reduction because PCA creates entirely new attributes, in addition to the fact that I wasn't sure how to visualize clusters with dimensions much greater than two. This would have opened more fronts on comparisons and interpretations on the effects of the different dimensionality reduction algorithms on clustering.

Finally, there were other hyperparameters I would've like to modify, such as the tolerance in ICA, and overall more runs with different seeds for the random projection. The error tolerance in ICA could have interesting impacts on the resulting attributes, since a smaller tolerance would supposedly yield to better results, albeit the running time would take much longer. Running random projections more times would make the results more robust and consistent, and would also improve the probability of finding a better projection direction to yield better classification accuracy.