Unsupervised Learning Analysis

Abstract

This paper will continue to explore machine learning techniques on the GOT and GGG datasets used in the previous assignments. Several algorithms for clustering and dimensionality reduction will be used to process the dataset and, finally, train a neural network. The first section of the paper will use k-means and expectation maximization to cluster the data. Next, the data will be run on dimensionality reduction algorithms: PCA, autoencoders, and randomized projections. We will then cluster the dimension reduced data. And finally, we will use both the dimensionality reduction and the clustering algorithms to train neural networks for both datasets.

Dataset

I will be reusing the 2 datasets I used in Assignment 1 that were obtained from Kaggle. These can be found in the input folder in the original file.

The first dataset is called “GOT.csv,” which contains information about various characters in the novels and HBO show, *Game of Thrones*. I will be using the features Prediction, Male, isNoble, isMarried and isPopular to determine if certain characters will survive. The prediction generated from the algorithms is compared to the target feature “isAlive” which is the correct answer. This is a classification problem. There are 1946 data points in total. All values are discrete and binary with 1 meaning true and 0 meaning false. There are no null points in the features. I selected this dataset because I thought it would be interesting to see if there are any data trends in the survival of GoT characters. I am a huge fan of the books and show, as are many students at Georgia Tech. I also thought it would be fascinating to see how the various machine learning algorithms predict the deaths of various important and popular characters.

The second dataset is called “GGG.csv.” It contains information about various monster traits and classifies those traits into monster types. The feature “color” was encoded into numbers to make for easier analysis with strictly numerical features. White is represented by 1, black is 2, blood is 3, blue is 4, clear is 5 and green is 6. The features bone_length, rotting_flesh, hair_length, has_soul will also be used to find the target type, which can be either a Ghost, Goblin or Ghoul. Similar to the feature ‘Color’, the original target type was a string categorical feature. I have changed the dataset to numerical classification to allow for easier analysis. Ghost is now represented by 1, Ghoul is 2 and Goblin is now 3. The attributes and the target feature are all continuous numbers. The dataset contains a total of 897 data points. This is a classification problem. I picked this dataset because it was one of the contests that Kaggle held in the past and I thought it would be an interesting dataset to train and test the algorithms on.

Clustering Algorithms

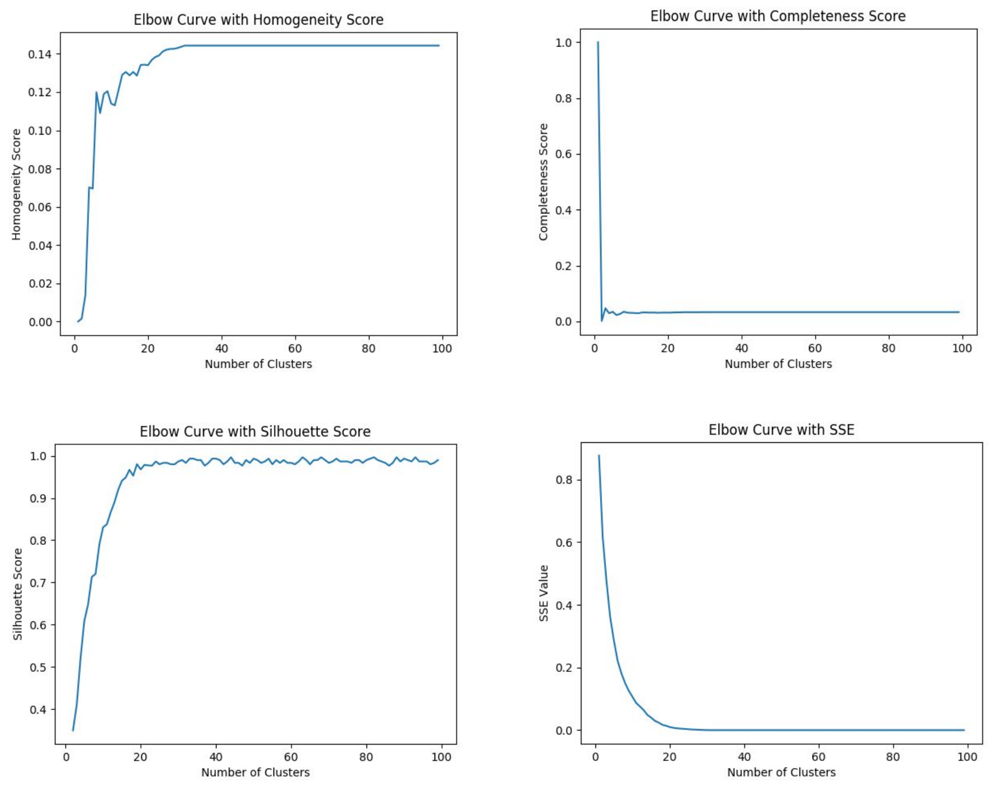
Clustering is used to group similar data points with the assumption that instances in the same cluster share similar qualities. Two clustering algorithms, k-Means clustering and expectation maximum, were applied on the GOT and GGG datasets. Both clustering algorithms were implemented using Scikit.learn.

k-Means:

The k-Means technique works by grouping the instances into different clusters by picking k random centers and classifying points around them in the same cluster. In each iteration, the center point is recalculated with the average location of all data points in the cluster to find a new centroid. This process is repeated until the centers do not move and convergence is reached. For the similarity/distance metric, Euclidean distance was used in both the GOT and GGG dataset. While Manhattan distance could be suitable for the GOT since it is not a continuous dataset, it would not be applicable to the GGG data points. Euclidean distance is a standard choice. Moreover, the datasets are not large in number of attributes, so other higher dimensionality distance measures are not suitable.

k-Means is tuned by altering the hyperparameter k representing the number of clusters. To find the best value, k-means models were scored by several clustering evaluation methods within a k value range from 1 to 100. Each model was evaluated with a homogeneity score, completeness score, silhouette score, and sum of squared error within each cluster (SSE) to find the best number of clusters. The homogeneity score describes how well each cluster contains only data points of a single class. Completeness score is similar in that it is a measure of how well all the data points of a given class are all part of the same cluster. Silhouette score measures how well each instance is related to a cluster in comparison to other clusters. Lastly, the SSE is a measure of variance within a cluster found by averaging the Euclidean distance of each instance within a cluster to the centroid. Figure 1 and 2 below show the different elbow graphs obtained for the GOT and GGG datasets respectively.

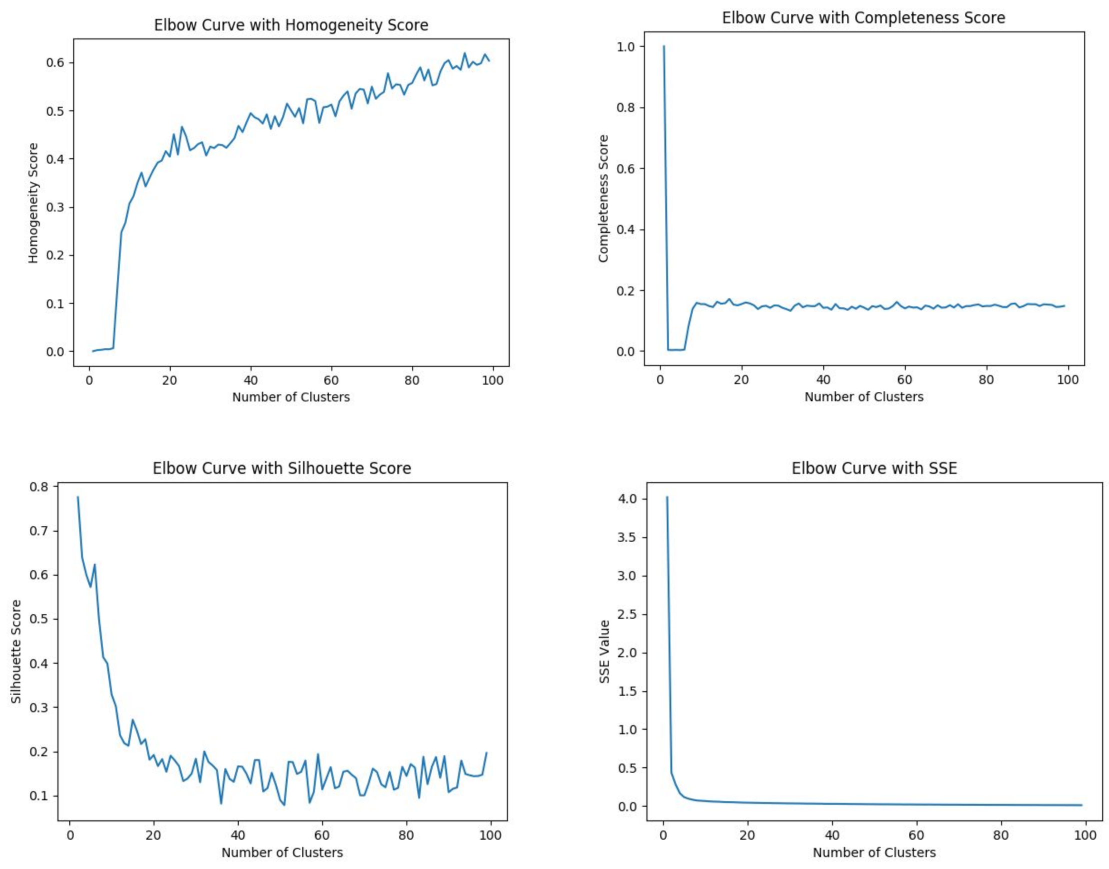
Figure 1. Elbow Curve Graphs for Clusterings of GOT Dataset



Using the elbow method, we can find the optimal number of clusters to choose for each dataset. For GOT, k = 4 seems to be the best choice according to the homogeneity curve. In the completeness graph, k = 2 is the lowest point before the graph plateaus. With the silhouette graph, the first angle occurs at k = 2, a more severe angle starts at k = 4, and the curve levels off completely after k = 15. In the SSE curve, the angle of the elbow seems to intensify around k = 4 and k = 6. We will use the recurring value of k = 2 as the best number of clusters for this dataset. This is expected as our domain knowledge suggests that there are only 2 classifications. The character either lives or dies. Moreover, the lower number of clusters will also combat overfitting. The lack of a clear elbow point in all four graphs suggests that the data might not be suited for clustering.

Figure 2 below shows the clustering evaluation graphs obtained for the GGG dataset. Using the elbow method as we did previously, we find that the optimal number of clusters for the GGG dataset is actually k = 3. While the homogeneity curve suggest that k = 6 may be the elbow point, there is a clear “descent to plateau” shift in the completeness, SSE and silhouette score curves at k = 3. Moreover, we also have the domain knowledge that the dataset has 3 classifications for Ghost, Ghouls and Goblins. So we will use k = 3 as the number of clusters for the GGG dataset.

Figure 2. Elbow Curve Graphs for Clusterings of GGG Dataset

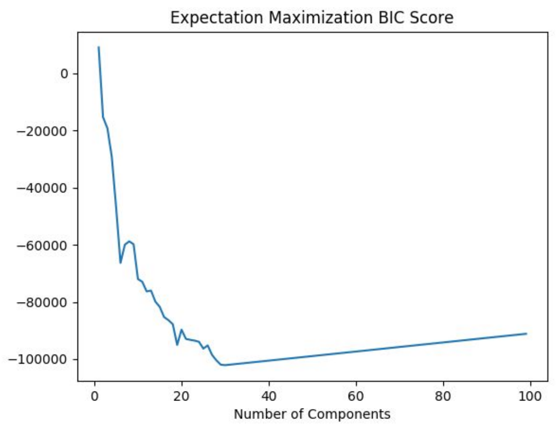


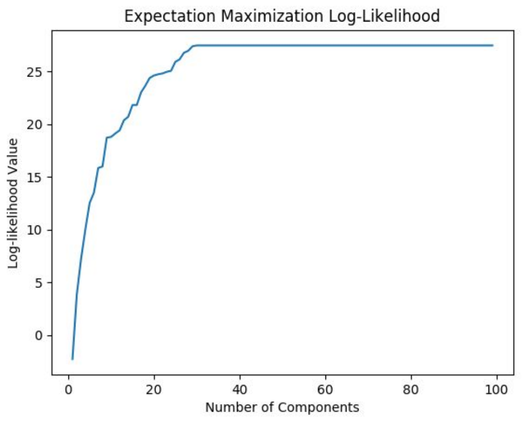
Expectation Maximization:

Expectation Maximization (EM) is another clustering technique that lets instances be a part of many clusters, as opposed to k-means clustering. In the k number of clusters, the algorithm calculates the probabilities for each point being in a particular cluster. This corresponds to a Gaussian distribution for each k cluster and EM tries to essentially ‘maximize’ the probability that the data came from this distribution of clusters and points at each iteration. Using Scikit.learn’s ‘Gaussian Mixture’ package, EM was implemented and run on both the GOT and GGG datasets. The clusters are evaluated using the log-likelihood of each current model. The Bayesian Information criteria (BIC) was also used to evaluate clusters and to help combat overfitting the data.

Figure 3 below shows the resulting graphs for the EM ran on the GOT dataset. From the Log-likelihood graph, we see that using 4 components provided a high value before it started to level off. Similarly, in the BIC curve, we see that k = 4 again provided the first elbow point before the curve started to level off. This result contrasts that of the k-Means graph, where we chose 2 clusters since lower values would prevent overfitting. We will continue to use k = 2 as this makes more sense to cluster the data in 2 groups. If we used k = 4, we would run into confusion on how to classify the 3rd and 4th cluster.

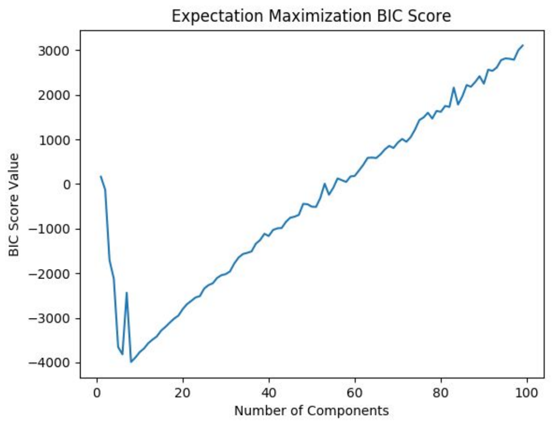
Figure 3. Evaluation Graphs for EM of GOT Dataset

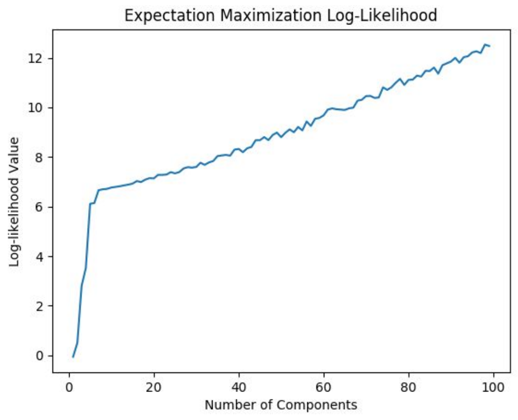




Similarly, Figure 4 shows the resulting graphs for the EM ran on the GGG dataset. With the Log-likelihood curve, we can see that the elbow occurs at a value of k = 3. The same value reoccurs in the BIC score curve. Using the same reasoning, we can see that a value of k = 3 is best choice for the number of clusters in the GGG dataset. As expected, this is the same trend we saw in the k-Means algorithm, from which we determined that k = 3 is the best fit.

Figure 4. Evaluation Graphs for EM of GGG Dataset

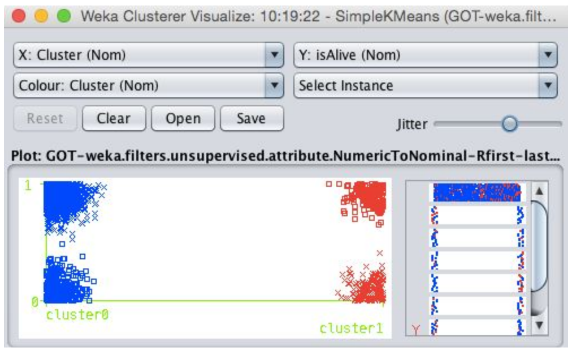




Clustering Conclusion:

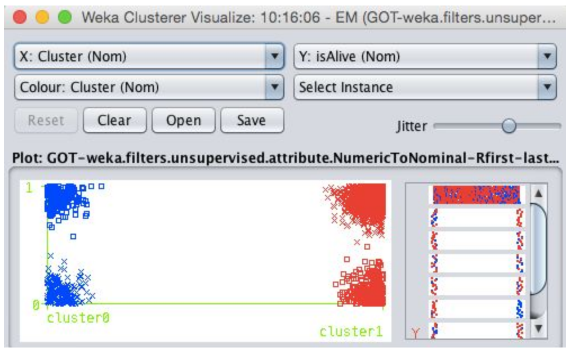
The clusterings of the GOT and GGG datasets are shown below in Figure 5 and 6 respectively. We used k = 2 for the k-Means Clustering and EM algorithm with the GOT dataset since there are 2 outcomes in the target attribute isAlive. Visualization of the clusters was obtained with WEKA software. We can see that there are still a lot of errors with the clusterings. With k-Means, 500 instances are incorrectly clusterer with a inaccuracy percentage of 25.6937 %. In EM, the clusterings are insignificantly better with 498 instances incorrectly clustered with an inaccuracy of 25.591%. There is only a slight improvement with EM. This could be because the use of probabilities might be better suited to clustering the GOT dataset. Moreover, the data set is not suited for the k-Means bias towards spherical clusters.

Figure 5. k-Means Clustering of GOT, k = 2 and EM Clustering of GOT, k = 2


Weka Clusterer Visualize: 10:19:22 - SimpleKMeans (GOT-weka.filt...

X: Cluster (Nom)
Y: isAlive (Nom)
Colour: Cluster (Nom)

Plot: GOT-weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last...

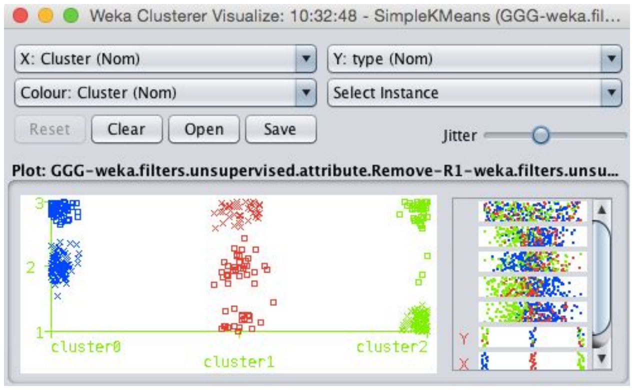

Weka Clusterer Visualize: 10:16:06 - EM (GOT-weka.filters.unsuper...

X: Cluster (Nom)
Y: isAlive (Nom)
Colour: Cluster (Nom)

Plot: GOT-weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last...

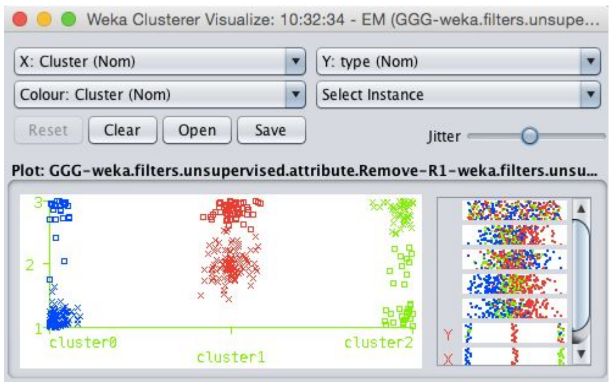
Similarly, in the GGG dataset, we use k = 3 because there are three classifications. The clustering graphs below were obtained from WEKA software. With k-Means, 138 instances were incorrectly clustered with an inaccuracy percentage of 37.1968%. EM clustering was better in comparison with 110 instances incorrectly and an inaccuracy percentage of 29.9191%. Again, the EM algorithm was able to cluster more efficiently than k-Means. This suggests that the features of GGG dataset is also not suited for the k-Means bias of spherical clusters. Moreover, soft clustering is a better technique for this dataset as opposed to the hard clustering used with k-Means.

Figure 6. K-Means Clustering of GGG, k = 3 and EM Clustering of GGG, k = 3


Weka Clusterer Visualize: 10:32:48 - SimpleKMeans (GGG-weka.fil...

X: Cluster (Nom)
Y: type (Nom)
Colour: Cluster (Nom)

Plot: GGG-weka.filters.unsupervised.attribute.Remove-R1-weka.filters.unsu...


Weka Clusterer Visualize: 10:32:34 - EM (GGG-weka.filters.unsupe...

X: Cluster (Nom)
Y: type (Nom)
Colour: Cluster (Nom)

Plot: GGG-weka.filters.unsupervised.attribute.Remove-R1-weka.filters.unsu...

Dimension Reduction Algorithms

In this section of the paper, we will run three dimension reducing algorithms: principal component analysis, autoencoders, and randomized projections. These algorithms work by transforming the given data into fewer dimensions. This is done to alleviate the curse of dimensionality by reducing the features needed to learn the target function. Simplifying the information improves both accuracy and runtime. Once the optimal number of components is found, the reduced data will be run again on the clustering algorithm mentioned in Section One to see how the clusters change with reduced dimensions.

Principal Component Analysis (PCA):

PCA is the first dimension reducing algorithm we will run on our two datasets. The technique works by finding principal component vectors (PCV), essentially orthogonal eigenvectors, to find a linear transformation of the input features that best explains the maximum variance. As more PCVs are added, the eigenvalue associated with the model decreases. Using Scikit.learn's PCA package, we ran the algorithm on both the datasets. The eigenvalues obtained for the GOT and GGG dataset are shown in Table 1 and 2 below. Both the GOT and GGG dataset have a maximum number of five attributes. For the GOT dataset, the eigenvalue for the 5th component was very small and could be removed in the classification of the data points. Similarly, in the GGG dataset, the eigenvalues became insignificant after the first component. The algorithm has interpreted that the remaining components add more noise to the data. However, we know that these attributes should not be ignored in further classification. The eigenvalues between the GOT dataset are a lot smaller than those from the GGG dataset. This suggests that there is an equal sparsity of information given in each direction of the components with the GOT dataset. On the other hand, component 1 has a lot of information about the in the GGG dataset. We will set the number of PCVs for the GOT dataset to 4. For GGG, we will use 1.

Table 1. Eigenvalues obtained for each Principal Component Vector in GOT

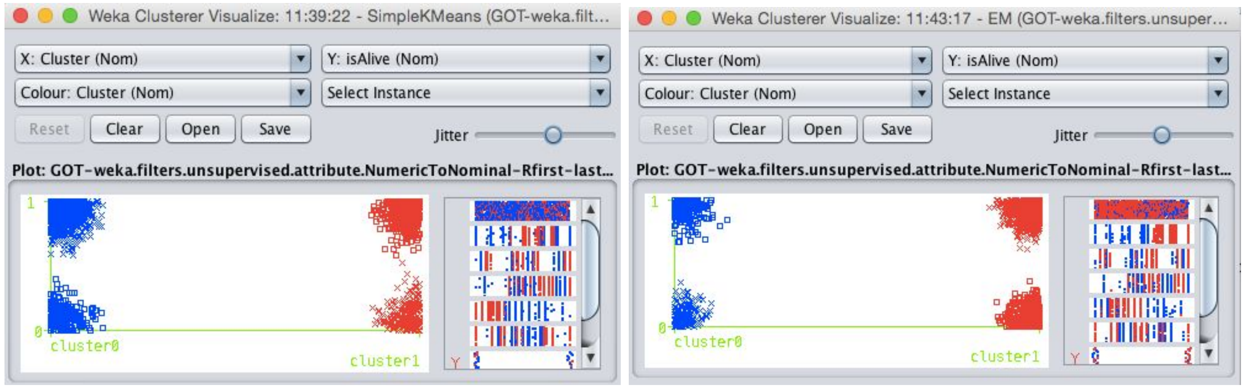
Principal component	1	2	3	4	5
Eigenvalue	0.31305874	0.21754507	0.19098999	0.10749708	0.04795067

Table 2. Eigenvalues obtained for each Principal Component Vector in GGG

Principal component	1	2	3	4	5
Eigenvalue	3.92674302	0.04993779	0.02084089	0.01514312	0.01274745

The dimensionally reduced data from the PCA was run with the clustering algorithms mentioned in Section One. The obtained clusterings using the PCA-reduced data are shown in Figure 7 and 8 below for the GOT and GGG dataset respectively. With the GOT dataset, we use k = 2 for the K-Means and EM clustering. With the k-Means reduced data clustering, 607 instances were incorrectly clustered with an inaccuracy rate of 31.1922%. This is worse than just using K-Means alone without the dimension reduction. EM does a better job with the reduced data with 522 incorrect clusterings and an inaccuracy percentage of 26.824%. But overall, EM without PCA does better in comparison. This is not unexpected since reducing the dimension does not always correlate to more accuracy. The generalizations might be too drastic and the original dataset is underrepresented.

Figure 7 K-Means (L) and EM (R) clustering with PCA reduced data, GOT dataset



Weka Clusterer Visualize: 11:39:22 - SimpleKMeans (GOT-weka.filt...

X: Cluster (Nom) Y: isAlive (Nom)
Colour: Cluster (Nom) Select Instance

Plot: GOT-weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last...

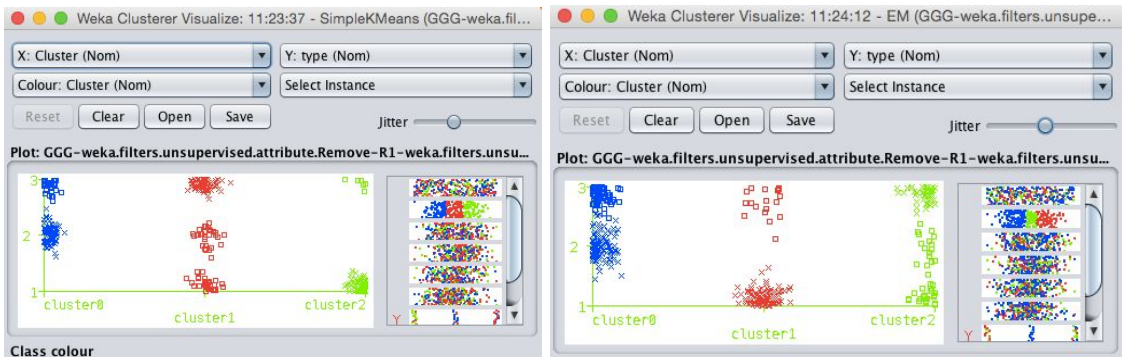
Weka Clusterer Visualize: 11:43:17 - EM (GOT-weka.filters.unsuper...

X: Cluster (Nom) Y: isAlive (Nom)
Colour: Cluster (Nom) Select Instance

Plot: GOT-weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last...

For the GGG dataset, the k-Means clustering with reduced data provides a better clustering than the original. There were 99 instances that were incorrectly clustered with an accuracy of 26.6846%. The PCA-reduced data was better able to simplify classification of the GGG dataset. The same cannot be said with the EM clustering as 119 instances were incorrectly clustered with an inaccuracy rate of 32.0755% However, this is only slightly worse the original EM clustering. The EM algorithm seemed to have trouble distinguishing ghouls and goblins with the PCA reduced data. In general, it seems that with K-means, reducing the dimensionality and selecting less features helped with clustering.

Figure 8. K-Means (L) and EM (R) clustering with PCA reduced data, GGG dataset



Weka Clusterer Visualize: 11:23:37 - SimpleKMeans (GGG-weka.fil...

X: Cluster (Nom) Y: type (Nom)
Colour: Cluster (Nom) Select Instance

Plot: GGG-weka.filters.unsupervised.attribute.Remove-R1-weka.filters.unsu...

Class colour

Weka Clusterer Visualize: 11:24:12 - EM (GGG-weka.filters.unsupe...

X: Cluster (Nom) Y: type (Nom)
Colour: Cluster (Nom) Select Instance

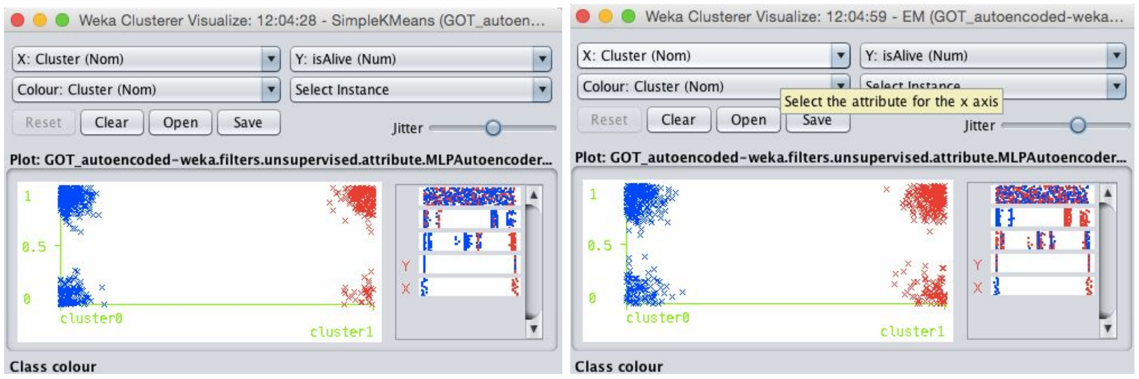
Plot: GGG-weka.filters.unsupervised.attribute.Remove-R1-weka.filters.unsu...

Autoencoders:

Autoencoders is the second dimension reducing algorithm we will run on our two datasets. These algorithms have two parts: the encoder and the decoder. The encoder first compresses the given input data into a lower dimensional representation. Then the decoder will reconstruct the original inputs based on the representation given by the encoder. This algorithm was implemented with WEKA on both datasets. Again, we run the clustering algorithms K-Means and EM on the dimensionally reduced dataset. We use the same k values as before. The clustering graphs are shown below in Figure 8 and 9.

The Autoencoder was run after standardizing the data. One hidden layer was used as this best minimized the squared error with weight decay. After running the autoencoder, the number of attributes was reduced to 3 for both the GGG and GOT dataset. Figure 8 below shows the clustering results from the Autoencoder reduced GOT data. The within-cluster sum of squared error was found to be 347.55 for the k-Means clustering. The EM clustering resulted in a log-likelihood value of 1.329. Both of these clusterings were worse than the original clustering without the dimension reduction. A similar outcome is seen in the GGG Dataset shown in Figure 9. The k-Means resulted in a within-cluster sum of squared error of 40.955. EM clustering had a log-likelihood value of -0.563.

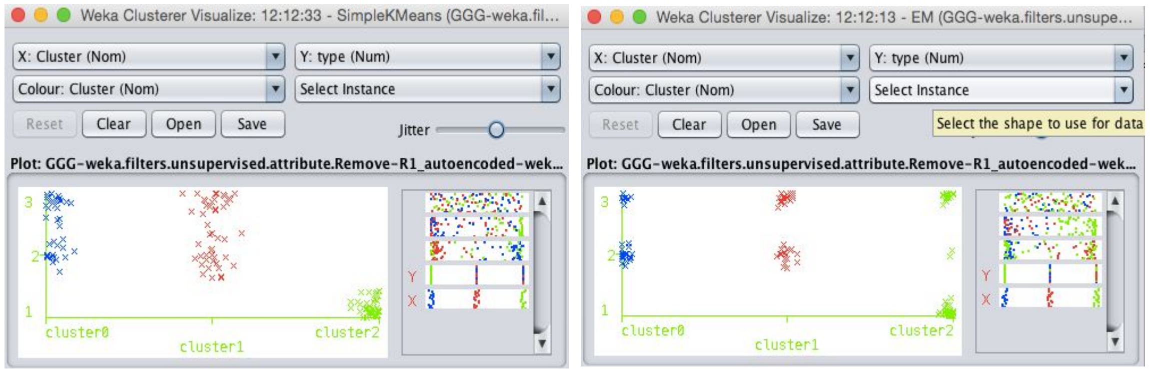
Figure 9. K-Means (L) and EM (R) clusters with Autoencoder reduced data, GOT Dataset



Weka Clusterer Visualize: 12:04:28 - SimpleKMeans (GOT_autoen...
X: Cluster (Nom) Y: isAlive (Num)
Colour: Cluster (Nom) Select Instance
Plot: GOT_autoencoded-weka.filters.unsupervised.attribute.MLPAutoencoder...

Weka Clusterer Visualize: 12:04:59 - EM (GOT_autoencoded-weka...
X: Cluster (Nom) Y: isAlive (Num)
Colour: Cluster (Nom) Select Instance
Select the attribute for the x axis
Plot: GOT_autoencoded-weka.filters.unsupervised.attribute.MLPAutoencoder...

Figure 10. K-Means (L) and EM (R) clusters with Autoencoder on GOT Dataset



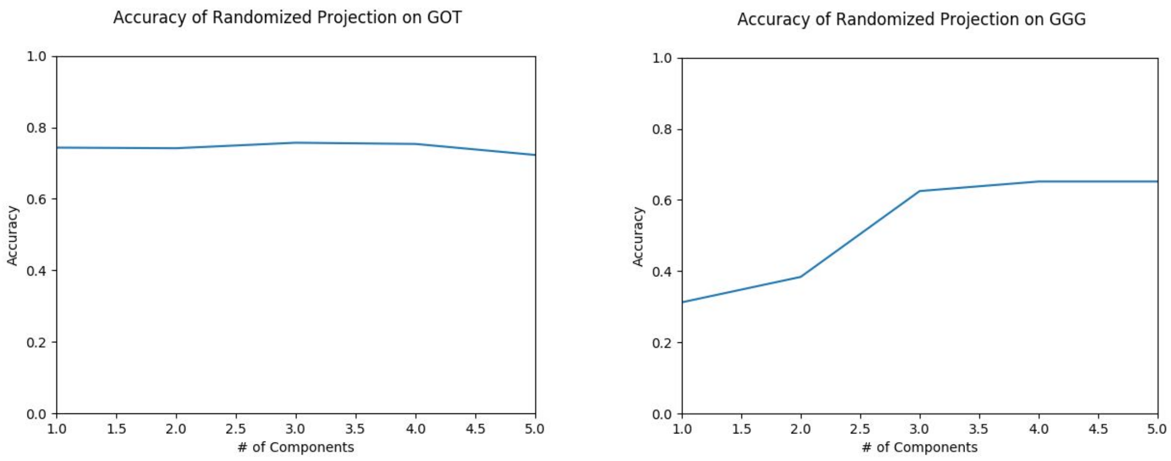
Weka Clusterer Visualize: 12:12:33 - SimpleKMeans (GGG-weka.fil...
X: Cluster (Nom) Y: type (Num)
Colour: Cluster (Nom) Select Instance
Plot: GGG-weka.filters.unsupervised.attribute.Remove-R1_autoencoded-wek...

Weka Clusterer Visualize: 12:12:13 - EM (GGG-weka.filters.unsupe...
X: Cluster (Nom) Y: type (Num)
Colour: Cluster (Nom) Select Instance
Select the shape to use for data
Plot: GGG-weka.filters.unsupervised.attribute.Remove-R1_autoencoded-wek...

Randomized Projections:

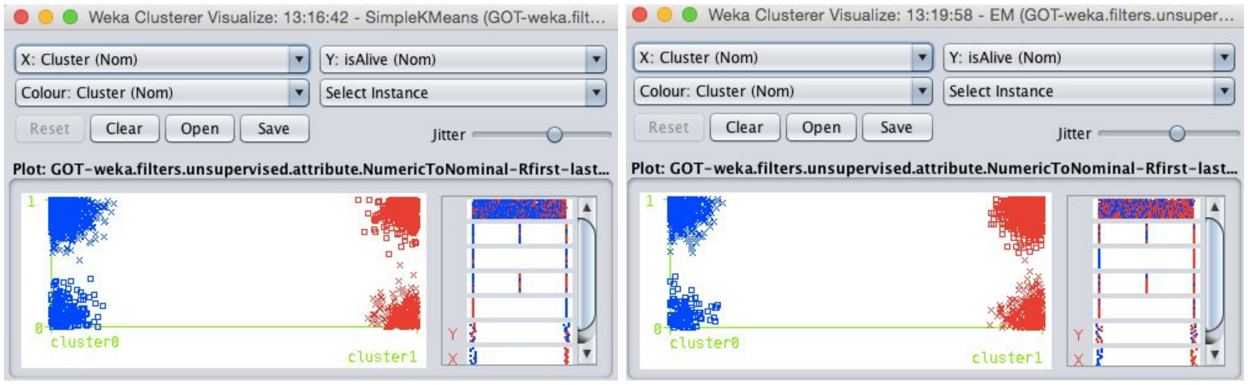
Randomized Projections (RP) is the last dimension reducing algorithm that we will run on the dataset. RP work by taking a target number of dimensions and projecting this input feature on a randomly generated Gaussian matrix in a smaller feature space. The hyperparameter that we need to tune is the target number of dimensions, which can be less or greater than the current dimensionality of the data. The GOT and GGG dataset were both run 10 times since the initial seed value greatly changes the results of the RP. Figure 10 below shows the relationship of tuning the target number of components to the accuracy of clustering the reduced data. For the GOT dataset, it seems that there is not much variation in accuracy with respect to the number of components used. We will use 4 as the number of components. For GGG, we will use 3 components.

Figure 11. Accuracy of Autoencoder reduced data on GOT and GGG dataset



The RP reduced data clustering performed on the K-Means and EM algorithms is shown in Figure 10. With K-Means, 514 instances were incorrectly clustered with an inaccuracy score of 26.4132%. With EM, there is a larger error rate of 49.6403% inaccuracy with 966 instances incorrectly clustered. The large difference could be due to the RP randomly choosing projected vectors, and therefore, there can be a lot of variance between the instances. The clustering of GOT with RP was better than the clusterings obtained with PCA. This could be due to the random projections allowing different exploration of feature spaces.

Figure 12. K-Means (L) and EM (R) clustering of RP reduced data, GOT dataset


Weka Clusterer Visualize: 13:16:42 - SimpleKMeans (GOT-weka.filt...

X: Cluster (Nom) Y: isAlive (Nom)
Colour: Cluster (Nom) Select Instance

Plot: GOT-weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last...

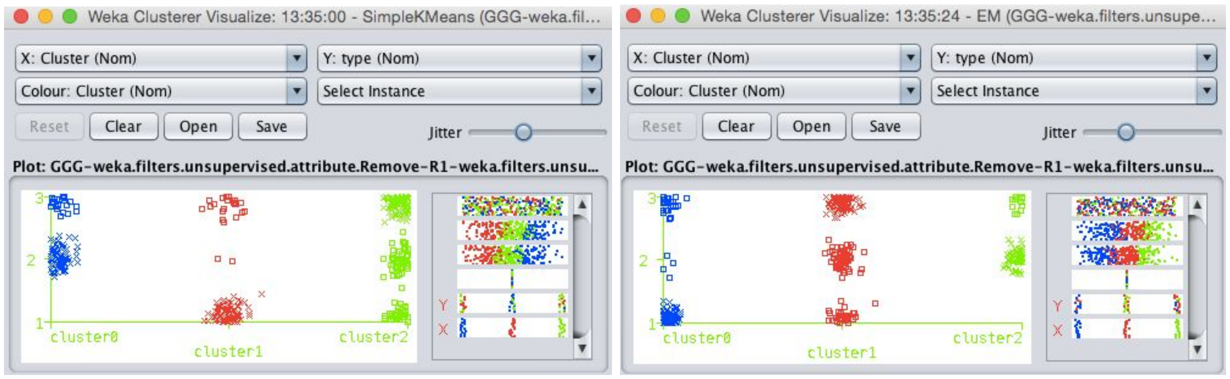
Weka Clusterer Visualize: 13:19:58 - EM (GOT-weka.filters.unsuper...

X: Cluster (Nom) Y: isAlive (Nom)
Colour: Cluster (Nom) Select Instance

Plot: GOT-weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last...

The GGG dataset performed better with the RP reduced dataset in comparison to clustering alone. In K-Means clustering, the algorithm incorrectly clustered 127 instances with an inaccuracy score 34.2318%. Similarly, in the EM clustering, the algorithm incorrectly clustered 122 instances with an inaccuracy score 32.8841%. With the GGG dataset, random projections performed slightly worse than PCA. However, RP perform a lot faster than PCA and can still obtain a similar performance to PCAs.

Figure 13. K-Means (L) and EM (R) clustering of RP reduced data, GGG dataset


Weka Clusterer Visualize: 13:35:00 - SimpleKMeans (GGG-weka.fil...

X: Cluster (Nom) Y: type (Nom)
Colour: Cluster (Nom) Select Instance

Plot: GGG-weka.filters.unsupervised.attribute.Remove-R1-weka.filters.unsu...

Weka Clusterer Visualize: 13:35:24 - EM (GGG-weka.filters.unsupe...

X: Cluster (Nom) Y: type (Nom)
Colour: Cluster (Nom) Select Instance

Plot: GGG-weka.filters.unsupervised.attribute.Remove-R1-weka.filters.unsu...

Dimension Reduction Conclusion

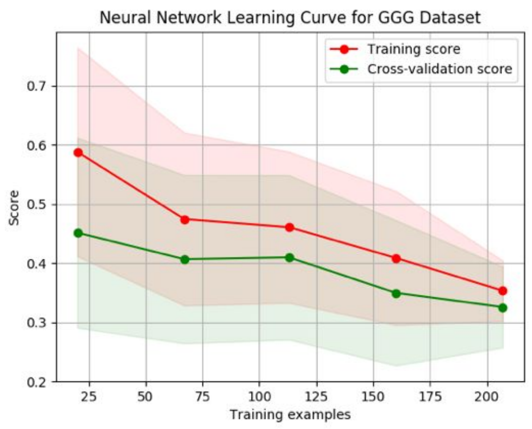
In general, dimension reduction algorithms were able to reduce the number of features needed for classification of the GOT and GGG dataset. They were also helpful in understanding which features are important in the classification. Overall, it seems that applying these algorithms generally improve the clustering of the dataset. However, this is not always a guarantee. We will now explore these algorithms by applying what we have learned about the data to reproduce and train the neural network from assignment 1.

Neural Networks

We will now explore the last part of this paper and use the reduced datasets to train and test the neural network we obtained in the first paper. We will specifically be talking about the GGG Neural Network created with a logistic activation function with 4 hidden layers. The learning curve for this neural net is shown in Figure 12. We will compare the results of the dimension reduction and clustering algorithms to this learning curve to see how classification in the GOT dataset is altered. The original testing accuracy score of this network was 77.3585 % with 0.54 seconds

We will train neural networks on the three dimensionality reduction algorithms with the GGG dataset. We will use the full dataset as training with a cross-validation of 10 sets. PCAs and RPs are implemented with scikit.learn whereas Autoencoders are implemented with WEKA.

Figure 14. Original NN Learning Curve on GGG dataset



Dimension Reduction on Neural Networks (NN):

The dimensionally reduced data from PCAs, Autoencoder and RP were used to train and test the Neural Network. The hidden layers was set to 4 for consistency. After training the neural network, the training accuracy and the time taken to build the model was recorded. Each model was built on WEKA. Table 3 below shows the results of the experiments.

Table 3. Dimension Reduction Dataset used to Train Neural Nets

Combination of NN	Training Accuracy %	Training Time (seconds)
PCA + NN	79.5148 %	0.25 seconds
Autoencoder + NN	73.5849 %	0.15 seconds
RP + NN	78.1671 %	0.24 seconds

In theory, dimensionally reducing the data should reduce the size of the network. This should also reduce the amount of data needed to train. We can see that autoencoders performed the worst in terms of training accuracy with respect to the original neural net. This could be due to the nature of the GGG dataset reduced with an autoencoder with 1 hidden level. The discriminative information to classify each monster type may lie in the low variance components making Autoencoders perform worse in this situation. Moreover, using less features to classify a small set of data such as GGG would be expected to perform less.

On the other hand, PCA and RP datasets helped the neural net achieve higher training accuracy by 2% and 0.8% respectively. Overall, these differences are negligible. It is worth mentioning that all three algorithms converged a lot faster than the original neural net. This is due to the fact that dimension reduction algorithms alleviate the curse of dimensionality and lets the model learn with less data faster.

Clustering on Neural Networks:

Lastly, the clustering algorithms from the dimensionally reduced dataset was then used to train and test the Neural Network. The hidden layers was set to 4. WEKA was used to first apply one of the three dimension reducing algorithms to the data. From there, the data was then clustered using the 'AddCluster' filter with either the k-Means or EM algorithm. Lastly, a neural net was built using the entire set to train the model. For each combination, the training accuracy and training time were recorded. The summary of the experiments is shown in table 3 in the next page.

We see that with PCA, k-Means slightly increased the training accuracy. EM actually made the reduced data harder to classify. This could be due to the fact that the probabilities are a lot more deterministic with less features in the data to cluster. This may have lead to more error with classification in the GGG dataset. The performance of the neural net with autoencoders improved with the help of both clustering algorithms. The clustering in both cases, helped the encoded data characterize the target feature when used with the neural network. Lastly, the randomized projection k-Means combination decreased the accuracy of the neural net. This could again be due to the random nature of RP where the initial seed can heavily skew the results. With EM clustering, the neural net did a lot better with an accuracy score of 79.2453%.

Table 3. Summary of clustering with Neural networks

Combination of NN	Training accuracy %	Training Time (sec)
PCA + Kmeans NN	79.7844 %	0.34 seconds
Autoencoder + Kmeans NN	74.6631 %	0.26 seconds
RP + Kmeans NN	76.8194 %	0.35 seconds
PCA + EM NN	77.628 %	0.32 seconds
Autoencoder + EM NN	74.124 %	0.23 seconds
RP + EM NN	79.2453 %	0.32 seconds

Neural Networks Conclusion:

Using simply dimension reduction algorithms, we found that PCAs achieved the highest training accuracy of 79.5148 % with the lowest time of 0.25 seconds. This is amazing as the neural net was actually able to find higher accuracy with half the time. With a combination of clustering and dimension reducing algorithms, we find that PCA and K-Means together achieve the best performance. The training accuracy was the highest achieved with a value of 79.7844 %. However, this is not always the case. We see that autoencoders perform a lot worse than the original neural network in all three implementations.

In conclusion, we find that there is a tradeoff with using dimension reducing algorithms to train neural nets. Selecting features, while they may help battle the curse of dimensionality, can also lead to less accuracy in classification. The algorithms end up choosing suboptimal features that may not generalize well. However, if the dataset we used to run this analysis was very large, using dimension reducing algorithms may have helped with runtime and simplifying neural networks. In general, they make classification problems a lot easier to train with less examples and iterations. They can achieve higher and competitive results in less time.